TUGAS AKHIR – KI141502

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN JUMLAH *ROUTING TABLE* DI VANETS

FAIQ FIRDAUSY
NRP 05111440000165

Dosen Pembimbing I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Ir. F.X. Arunanto, M.Sc.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN JUMLAH *ROUTING TABLE* DI VANETS

FAIQ FIRDAUSY
NRP 05111440000165

Dosen Pembimbing I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Ir. F.X. Arunanto, M.Sc.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

MODIFICATION OF FORWARDING NODE SELECTION FOR AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON AMOUNT OF ROUTING TABLE IN VANETS

FAIQ FIRDAUSY
NRP 05111440000165

First Advisor
Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

Second Advisor
Ir. F.X. Arunanto, M.Sc.

Department of Informatics
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR (AODV)* BERDASARKAN JUMLAH *ROUTING TABLE* DI VANETS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur Jaringan Komputer
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
FAIQ FIRDAUSY
NRP: 05111440000165

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
   (NIP. 198410162008121002) (Pembimbing 1)

2. Ir. F.X Arunanto, M.Sc. .....
   (NIP. 195701011983031004) (Pembimbing 2)

SURABAYA
JUNI, 2018

(Halaman ini sengaja dikosongkan)

MODIFIKASI PEMILIHAN *FORWARDING NODE* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN JUMLAH *ROUTING TABLE* DI VANETS

Nama Mahasiswa	**: Faiq Firdausy**
NRP	**: 05111440000165**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Ir. F.X Arunanto, M.Sc.**

Abstrak

Vehicular Ad hoc Networks (VANETs) merupakan salah satu teknologi yang banyak dikembangkan di berbagai Negara. Fokus utama penelitian VANETs adalah *routing protocol*. Ada banyak *routing protocol* yang dapat diimplementasikan pada VANETs salah satunya adalah *Ad-Hoc On Demand Distance Vector* (AODV). AODV merupakan salah satu *routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*. Sebuah *routing protocol* yang hanya akan membuat rute ketika ada paket yang ingin dikirim.

Modifikasi akan dilakukan pada proses pengiriman paket *route request* (RREQ), yaitu dengan cara menentukan jumlah *neighbor node* yang bertugas mengirim ulang (*rebroadcast*) paket RREQ. Hal ini dilakukan dengan cara melihat jumlah *routing table* dan *neighbor node* dari tiap *node* tersebut, lalu jika *node* tersebut memiliki jumlah *routing table* dan *neighbor node* kurang dari jumlah *threshold*, *node* tersebut menjadi *forwarding node* dan *node* tersebut yang bisa melakukan proses *rebroadcast*. Jika paket RREQ sampai pada *node* tujuan, maka *node* tujuan akan mengirim paket *route reply* (RREP) ke *node* asal. Lalu rute untuk pengiriman paket akan terbentuk. Modifikasi yang dilakukan akan menghasilkan *routing overhead* dan *forwarded route request* yang lebih kecil daripada *routing protocol* AODV yang asli.

Pada tugas akhir ini, performa pada *routing protocol* AODV yang telah dimodifikasi menghasilkan performa yang lebih bagus. Dibuktikan dengan skenario *real* yang menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 3.37%, rata-rata penurunan *routing overhead* sebesar 7.14%, rata rata penurunan end to end delay sebesar 36.64%, dan juga rata-rata penurunan *forwarded route request* sebesar 58.79%.

Kata kunci: AODV, *Forwarding Node, Threshold,* VANETs

MODIFICATION OF FORWARDING NODE SELECTION FOR AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON AMOUNT OF ROUTING TABLE IN VANETS

Student's Name	**:**	**Faiq Firdausy**
Student's ID	**:**	**05111440000165**
Department	**:**	**Informatika FTIK-ITS**
First Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.**
Second Advisor	**:**	**Ir. F.X Arunanto, M.Sc.**

Abstract

Vehicular Ad Hoc Network (VANET) is one technology that has been developed in various countries. The main focus of research VANET is a routing protocol efficiency. There are many routing protocol on VANETs, one of them is Ad-Hoc On Demand Distance Vector (AODV). AODV is classified to reactive routing protocol. A routing protocol that only make the route when there are pakets to be sent.

In this thesis a solution have been made by modifying the delivery of route request (RREQ) process, which is determine the neighboring node that assigned to rebroadcast RREQ packet. By looking at the number of neighbor node and the number of routing table of each node, and then if the number of neighbor nodes and the number of routing table less than threshold, the node becomes a forwarding node and the node will rebroadcast the packet. If the RREQ packet has reached the destination node, the destination node will send back a route reply (RREP) packet to the source node and the route will be formed on cache.

The performance of the modified protocol has a better result than the original routing protocol AODV.. It is proven that in real scenario, there is an enhancement of average packet delivery ratio by 3.37%, reduction of average routing overhead by 7.14%, reduction of average end to end delay by 36.64% and reduction of average forwarded route request by 58.79%

Keyword: AODV, Forwarding Node, Threshold, VANETs

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul

"Modifikasi Pemilihan *Forwarding Node* Pada Ad-Hoc On Demand Distance Vector (AODV) Berdasarkan Jumlah *Routing Table* Di VANETS"

Harapan dari penulis semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT.
2. Keluarga penulis (Mama, Papa, Mas Aby dan keluarga penulis yang lain) yang selalu memberikan dukungan baik berupa doa, moral, dan material yang tak terhingga kepada penulis, sehingga penulis dapat menyelesaikan Tugas Akhir ini.
3. Bapak Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. dan Bapak Ir. F.X Arunanto, M.Sc. selaku dosen pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan Tugas Akhir ini.
4. Bapak Dr.Eng. Darlis Herumurti, S.Kom., M.Kom. selaku kepala Departemen Informatika ITS.
5. Teman-teman dari Warkop x Jojoran (Afif, Faris, Aldo, Nezar, Sani, Botak, Lian, Buyung, Paul, Pentol, Oing, Dyo, Hilman, Fikry, Hamka, Faiq, Tras, Cimeng, Hari,

Hakim, Upil, Rage, Fito, Anandi, Amik, Bajikas, Sekbay, Penyok, Kevin, Riefqy, Akhyar, Nanda, Dito, Rian, Nanda, Fathur, dan Petrus) yang selalu memberikan semangat, selalu memberikan hiburan kepada penulis, teman-teman yang sering diajak nongkrong, teman-teman yang bisa diajak untuk bertukar pikiran dan pendapat, dan juga menjadi keluarga baru penulis saat berkuliah di Departemen Informatika ITS.

6. Sahabat-sahabat penulis di angkatan 2014 (Vivi, Kania, Bebet, Nay, Raras, Tiara, Pina, Anindita, Nafia dan sahabat 2014 yang lain) yang telah menemani dan berjuang bersama penulis selama berkuliah di Departemen Informatika ITS.
7. Sahabat penulis (Andofa, Eko, Andi, Nafis, Alda, Amadea, dan sahabat penulis lain yang tidak dapat disebutkan satu per satu) yang selalu membantu, menghibur, menjadi tempat bertukar ilmu, serta berjuang bersama-sama dengan penulis.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Semoga untuk semua orang dan terutama yang sudah membaca buku ini, dimanapun, kapanpun, dan bagaimanapun kondisinya penulis doakan selalu bahagia. Aamiin.

Surabaya, 16 Mei 2018

Faiq Firdausy

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di zaman yang serba modern ini semakin hari jumlah kendaraan di jalan raya semakin meningkat. Sekitar 1,2 juta orang tewas setiap tahun karena kecelakaan di lalu lintas. [1] Keamanan lalu lintas, telah menjadi masalah yang menantang untuk diselesaikan. Salah satu solusi yang dapat digunakan yaitu menyediakan informasi lalu lintas pada setiap pengguna lalu lintas, sehingga pengguna jalan dapat menggunakannya untuk menganalisa kondisi lalu lintas. Setiap kendaraan dijalan raya bergerak dengan cepat, oleh karena itu jaringan seluler yang mampu beroperasi tanpa dukungan infrastruktur dibutuhkan. Untuk menjawab tantangan tersebut kita dapat memanfaatkan teknologi jaringan *Ad-Hoc* yang mana mendasari pembuatan *Vehicular Ad-Hoc Networks* (VANETs).

VANETs adalah jaringan *multihop* tanpa infrastruktur tetap, terdiri dari kendaraan yang bergerak berkomunikasi satu sama lain [2]. VANETs merupakan pengembangan dari *Mobile Ad-Hoc Networks* (MANETs). Implementasi dari MANETs telah menciptakan beberapa *routing protocol*. Berdasarkan perlakuannya terhadap rute, *routing protocol* dalam MANETs dibedakan menjadi dua, yaitu *routing protocol* bersifat proaktif dan *routing protocol* bersifat reaktif. *Routing protocol* proaktif menggunakan basis data untuk menyimpan *node* dan rute yang berada dalam jaringan contohnya adalah *Optimized Link State Routing Protocol* (OLSR), sedangkan *routing protocol* reaktif tidak menggunakan basis data untuk membentuk sebuah rute. Contohnya adalah *Ad hoc On demand Distance Vector* (AODV).

Salah satu tantangan utama dalam VANET adalah untuk mencari rute yang efisien untuk mengirim data dari *source* ke *destination*. [2] Karena perubahan topologi yang sangat cepat membuat sulit untuk merancang protokol routing yang efisien VANETs [3]. Beberapa studi perbandingan telah menyarankan *Ad hoc On demand Distance Vector* (AODV), protokol MANET yang terkenal adaptif terhadap perubahan dinamis pada jaringan untuk menjadi kandidat terbaik untuk digunakan pada VANETs. [4] Tetapi terdapat suatu masalah yang menyebabkan AODV belum bisa di gunakan secara optimal pada VANETs yaitu pada proses route discovery, setiap node yang menerima RREQ akan re-broadcast paket sehingga membuat jaringan overload

Pada Tugas Akhir ini diusulkan suatu mekanisme *routing discovery* pada *reactive routing* AODV untuk memperoleh rute berdasarkan jumlah *routing table* pada VANETs. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi Packet Delivery Ratio (PDR), Routing Overhead, End-to-End Delay, dan Forwarded Route Request (RREQ F).

1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah pada Tugas Akhir ini:

1. Bagaimana membatasi jumlah *forwarding node* dalam proses *rebroadcast route request* (RREQ)?
2. Bagaimana dampak pembatasan jumlah *forwarding node* terhadap performa *routing protocol* AODV secara keseluruhan?

1.3 Batasan Permasalahan

Batasan masalah pada Tugas Akhir ini adalah sebagai berikut:

1. Jaringan yang digunakan adalah VANETs.

2. Routing protocol yang diujicobakan yaitu AODV.
3. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2).
4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah untuk mereduksi jumlah *neighbor node* yang bertanggungjawab untuk *rebroadcast* RREQ *packet* sehingga dapat mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan pada *routing protocol* AODV.

1.5 Manfaat

Dengan dibuatnya Tugas Akhir ini dapat memberikan informasi tentang dampak dari pembatasan *forwarding node* berdasarkan jumlah *routing table* terhadap kinerja *routing protocol* AODV di lingkungan VANETs.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir berisi pendahuluan, deskripsi, dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, manfaat dan tujuan dari hasil pembuatan Tugas Akhir ini. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung

pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada studi literatur ini, akan dipelajari sejumlah referensi yang diperlukan dalam pengerjaan Tugas Akhir ini yaitu mengenai VANETs, AODV, NS-2, OpenStreetMap, Java OpenStreetMap, *Simulation of Urban Mobility* (SUMO), dan AWK.

1.6.3 Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal Tugas Akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan NS-2 sebagai *Network Simulator*, bahasa C/C++ sebagai bahasa pemrograman, SUMO, dan JOSM sebagai perangkat lunak untuk uji coba mengimplementasikan metode yang sudah diajukan.

1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan dengan VANETs *simulator generator* dan *traffic generator* yaitu SUMO untuk membuat simulasi keadaan topologi untuk diujikan. Kemudian simulasi yang dibuat pada SUMO tersebut dijalankan pada NS-2 *Network Simulator* dan akan menghasilkan *trace file*. Dari *trace file* tersebut akan dihitung *packet delivery ratio* (PDR), *end-to-end delay* (E2E), *routing overhead* (RO), dan *forwarded route request* (RREQ F) untuk mengetahui performa *routing protocol* yang telah dimodifikasi.

1.6.5 Penyusunan Buku

Pada tahap ini disusun buku sebagai dokumentasi dari pelaksanaan Tugas Akhir yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai VANETs, AODV, NS-2, OpenStreetMap, Java OpenStreetMap Editor, SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS-2, perancangan modifikasi AODV, serta perancangan metrik analisis (*Packet Delivery Ratio*, *End-to-End Delay*, *Routing Overhead*, dan *Forwarded Route Request*)
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, pembuatan peta untuk skenario *grid* dan skenario *real* menggunakan OpenStreetMap dan SUMO, melakukan simulasi menggunakan NS-2, dan perhitungan metrik analisis.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi modifikasi pada *routing protocol* AODV yang telah dilakukan

untuk menyelesaikan masalah yang dibahas pada Tugas Akhir. Pengujian dilakukan dengan skenario yang di*generate* oleh SUMO dan dijalankan di NS-2 untuk mendapatkan data uji PDR, E2E, RO, dan RREQ F yang nantinya akan dianalisis menggunakan skrip awk dan dilakukan perbandingan performa *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi.

6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

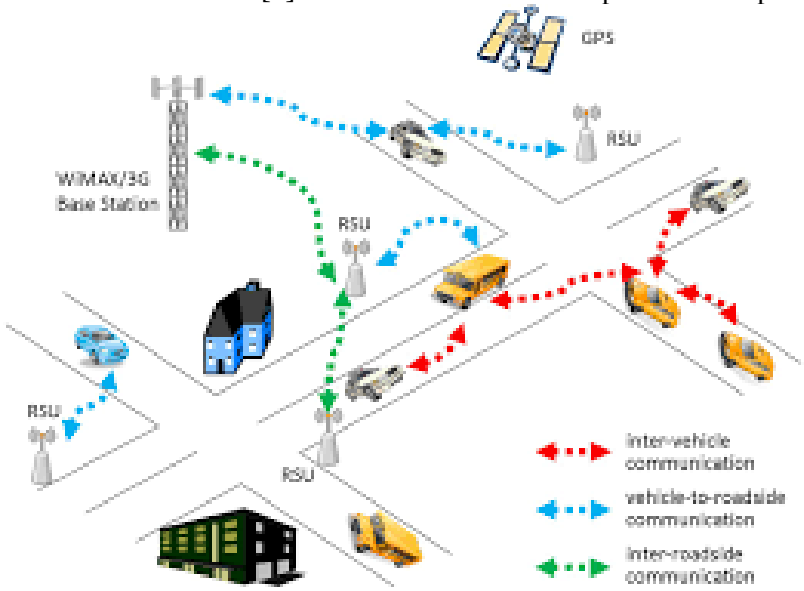
BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 VANETs

VANETs adalah salah satu pengaplikasian dari jaringan *Ad Hoc Mobile.* [1] Sebuah jaringan terorganisir yang dibentuk dengan menghubungkan kendaraan dan RSU (*Roadside Unit*) dan RSU lebih lanjut terhubung ke jaringan *backbone* berkecepatan tinggi melalui koneksi jaringan. Kepentingan peningkatan baru-baru ini telah diajukan pada aplikasi melalui V2V (*Vehicle to Vehicle*) dan V2I (*Vehicle to Infrastructure*) komunikasi, bertujuan untuk meningkatkan keselamatan mengemudi dan manajemen lalu lintas sementara menyediakan *driver* dan penumpang dengan akses Internet. Dalam VANETs, RSUs dapat memberikan bantuan dalam menemukan fasilitas seperti restoran dan pompa bensin, dan membroadcast pesan yang terkait seperti (maksimum kurva kecepatan) pemberitahuan untuk memberikan pengendara informasi. Sebagai contoh, sebuah kendaraan dapat berkomunikasi dengan lampu lalu lintas cahaya melalui V2I komunikasi, dan lampu lalu lintas dapat menunjukkan ke kendaraan ketika keadaan lampu ke kuning atau merah. Ini dapat berfungsi sebagai tanda pemberitahuan kepada pengemudi, dan akan sangat membantu para pengendara ketika mereka sedang berkendara selama kondisi cuaca musim dingin atau di daerah asing. Hal ini dapat mengurangi terjadinya kecelakaan. Melalui komunikasi V2V, pengendara bisa mendapatkan informasi yang lebih baik dan mengambil tindakan awal untuk menanggapi situasi yang abnormal. Untuk mencapai hal ini, suatu OBU (*On Board Unit*) secara teratur menyiarkan pesan yang terkait dengan informasi dari posisi pengendara, waktu saat ini, arah mengemudi, kecepatan,

status rem, sudut kemudi, lampu sen, percepatan / perlambatan, kondisi lalu lintas [5]. Ilustrasi VANETs dapat dilihat pada



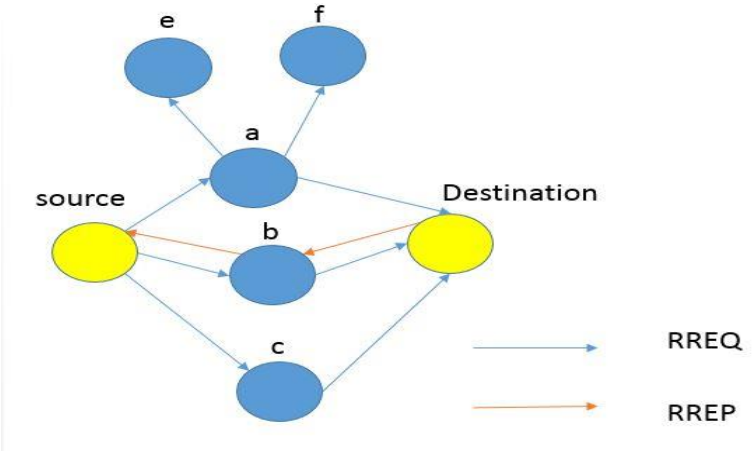
Gambar 2.1 Gambar ilustrasi VANETs

2.2 *Ad-hoc On demand Distance Vector (AODV)*

Ad-hoc On demand Distance Vector (AODV) adalah salah satu routing protokol yang termasuk dalam klasifikasi reactive routing protocol. Sebuah protokol yang hanya membuat sebuah rute saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561.

AODV mampu melakukan pengiriman data dengan metode *Multicast* dan Unicast. AODV akan mengirimkan *route request* (RREQ) dengan metode *Multicast* yaitu dengan mengirimkan RREQ ke semua *node* hingga *source node* menemukan *destination node.* Lalu route reply (RREP) akan dikirimkan oleh *destination node*

menuju *source node* dengan metode *unicast.* Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2 .



Gambar 2.2 Ilustrasi AODV

Sebagai contoh proses *route discovery* dalam AODV, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *source node*, mencari rute untuk menuju *destination node*. *Source node* akan membuat paket RREQ dan melakukan broadcast kepada semua *node* tetangganya *(neighbor node).* Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari yang ada pada *routing table* dan rute menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan *(drop).* Jika d*estination sequence number* pada RREQ lebih besar dibandingkan dengan yang terdapat pada *routing table*, maka entry pada *routing table* akan diperbarui dan paket tersebut akan diteruskan oleh *neighbor node* sekaligus membuat *reverse path* menuju *source node*. Paket RREQ akan diteruskan hingga mencapai *node* F. Kemudian, jika rute menuju node F sudah terbentuk di dalam routing table dan memiliki *routing flags "up",* maka *node* F akan mengirimkan paket RREP melalui rute tersebut menuju *node* .

Ciri utama dari AODV adalah adanya *routing table*. Informasi pada *routing table* harus tetap disimpan walaupun rute itu hanya dapat digunakan sementara. [4] *routing table* tersebut berisi field sebagai berikut:

- Destination Address: berisi alamat dari node tujuan.
- Destination Sequence Number: sequence number dari jalur komunikasi.
- Next Hop: alamat node yang akan meneruskan paket data.
- Hop Count: jumlah hop yang harus dilakukan agar paket dapat mencapai node tujuan
- Lifetime: waktu dalam milidetik yang diperlukan node untuk menerima RREP.
- Routing Flags: status jalur. Terdapat tiga tipe status, yaitu up (valid), down (tidak valid) atau sedang diperbaiki

Pada Tugas Akhir ini, penulis menggunakan AODV sebagai *routing protocol* yang akan dimodifikasi. Penulis akan membandingkan performa *routing protocol* AODV asli dengan *routing protocol* AODV yang dimodifikasi.

2.3 *Network Simulator-2* (NS-2)

Network Simulator (Versi 2), yang banyak dikenal dengan NS-2, adalah sebuah alat simulasi berbasis aktivitas yang berguna dalam mempelajari sifat dinamis jaringan komunikasi. Simulasi fungsi jaringan kabel, nirkabel, dan protokol dapat dilakukan dengan menggunakan NS-2. NS-2 menggunakan dua bahasa utama yaitu Bahasa C++ dan *Object-oriented Tool Command Language* (OTCL). Di NS-2 mendefinisikan mekanisme internal (*backend*) dari objek simulasi, dan OTCL mendefinisikan lingkungan simulasi eksternal (*frontend*) untuk perakitan dan konfigurasi objek. Setelah simulasi, NS-2 memberikan output simulasi baik dalam bentuk file NAM atau *trace file* [6].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan *routing protocol* AODV

yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan sebagai informasi untuk mengukur performa *routing protocol* AODV yang sudah dimodifikasi.

2.3.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus sudah ter*install* sebelum memulai instalasi NS-2. Untuk meng*install dependency* yang dibutuhkan dapat dilakukan dengan *command* yang ditunjukkan perintah berikut:

```
sudo apt-get install build-essential autoconf
automake libxmu-dev
```

Setelah meng*install dependency* yang dibutuhkan, ekstrak *package* NS-2 dan ubah baris kode ke-137 pada *file* ls.h di folder linkstate menjadi seperti perintah berikut:

```
void eraseAll(){this->erase(baseMap::begin(),
baseMap::end()); }
```

Install NS-2 dengan menjalankan perintah ./install pada folder NS-2.

2.3.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang disimulasikan. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.1

Tabel 2.1 Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi
1	Event	s: *sent*

Kolom ke-	Penjelasan	Isi
		r: *received* f: *forwarded* D: *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_: dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT: *application* RTR: *routing* LL: *link layer* IFQ: *packet queue* MAC: MAC PHY: *physical*
5	*Flag*	---: Tidak ada
6	*Sequence Number*	Nomor paket
7	Tipe Paket	AODV: paket *routing AODV* Cbr: berkas paket CBR (*Constant Bit Rate*) RTS: *Request To Send* yang dihasilkan MAC 802.11 CTS: *Clear To Send* yang dihasilkan MAC 802.11 ACK: MAC ACK ARP: Paket *link layer Address Resolution Protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a: perkiraan waktu paket b: alamat penerima c: alamat asal d: IP *header*
10	*Flag*	------: Tidak ada

Kolom ke-	Penjelasan	Isi
11	Detail IP *source*, *destination*, dan *nexthop*	[a:b c:d e f] a: IP *source node* b: *port source node* c: IP *destination node* (jika -1 berarti *broadcast*) d: *port destination node* e: IP *header* ttl f: IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast)*

2.4 OpenStreetMap

OpenStreetMap merupakan proyek kolaboratif untuk membuat sebuah peta dunia yang dapat dengan bebas disunting oleh siapapun. OpenStreetMap digunakan sebagai data peta pada berbagai *website*, aplikasi *mobile*, dan *hardware device*. OpenStreetMap dibangun oleh komunitas pembuat peta yang berkontribusi dan mengelola data mengenai jalan raya, kafe, stasiun kereta api, dan sebagainya di seluruh dunia [7].

Pada Tugas Akhir ini, penulis menggunakan data yang tersedia pada OSM untuk membuat skenario lalu lintas berdasarkan peta daerah di Surabaya. Peta yang diambil lalu digunakan untuk simulasi skenario *real* VANETs.

2.5 Java OpenStreetMap Editor (JOSM)

Java OpenStreetMap Editor (JOSM) adalah aplikasi yang dikembangkan oleh Immanuel Scholz. Aplikasi JOSM untuk menyunting data yang didapatkan dari OpenStreetMap. Aplikasi JOSM dapat diunduh pada halaman https://josm.openstreetmap.de. Penulis menggunakan aplikasi ini untuk menyunting dan merapikan peta yang diunduh dari OpenStreetMap [8].

Pada Tugas Akhir ini, penulis menggunakan data yang telah diambil dari OSM lalu disunting. Map yang disunting yaitu menghilangkan dan juga menyambungkan jalan yang ada, dan menghilangkan gedung-gedung yang ada di map.

2.6 Simulation of Urban Mobility (SUMO)

Simulation of Urban Mobility atau disingkat SUMO merupakan simulasi lalu lintas jalan raya yang *open source*, *microscopic*, dan *multi-modal*. SUMO mensimulasikan bagaimana permintaan lalu lintas yang terdiri dari beberapa kendaraan berjalan pada jalan raya yang telah ditentukan. Simulasi SUMO dapat menunjukkan beberapa topik manajemen lalu lintas dalam skala besar. SUMO murni *microscopic*, yang artinya setiap kendaraan dimodelkan secara eksplisit, memiliki rutenya sendiri, dan bergerak secara individu dalam jaringan [9].

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk men*generate* skenario VANETs, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan sebenarnya lalu lintas yang ingin disimulasikan. Untuk setiap *generate* skenario VANETs menggunakan SUMO akan menghasilkan pergerakan node yang acak sehingga tidak akan sama setiap *generate* menggunakan SUMO.

2.7 AWK

AWK merupakan sebuah bahasa pemrograman yang didesain untuk *text-processing* dan biasanya digunakan sebagai alat ekstrasi data dan pelaporan. AWK bersifat *data-driven* yang berisikan kumpulan perintah yang akan dijalankan pada data tekstural baik secara langsung pada *file* atau digunakan sebagai bagian dari *pipeline* [10].

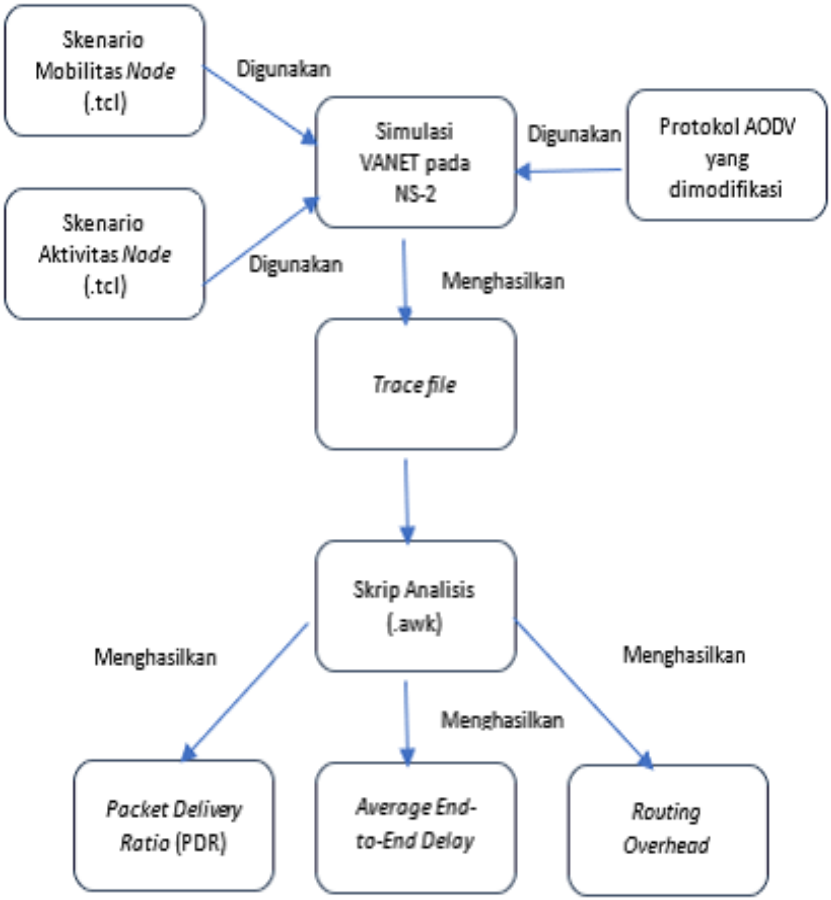
BAB III
PERANCANGAN

Bab ini membahas mengenai perancangan implementasi sistem yang dibuat pada Tugas Akhir. Bagian yang akan dijelaskan pada bab ini berawal dari deskripsi umum, perancangan skenario, hingga alur dan implementasinya.

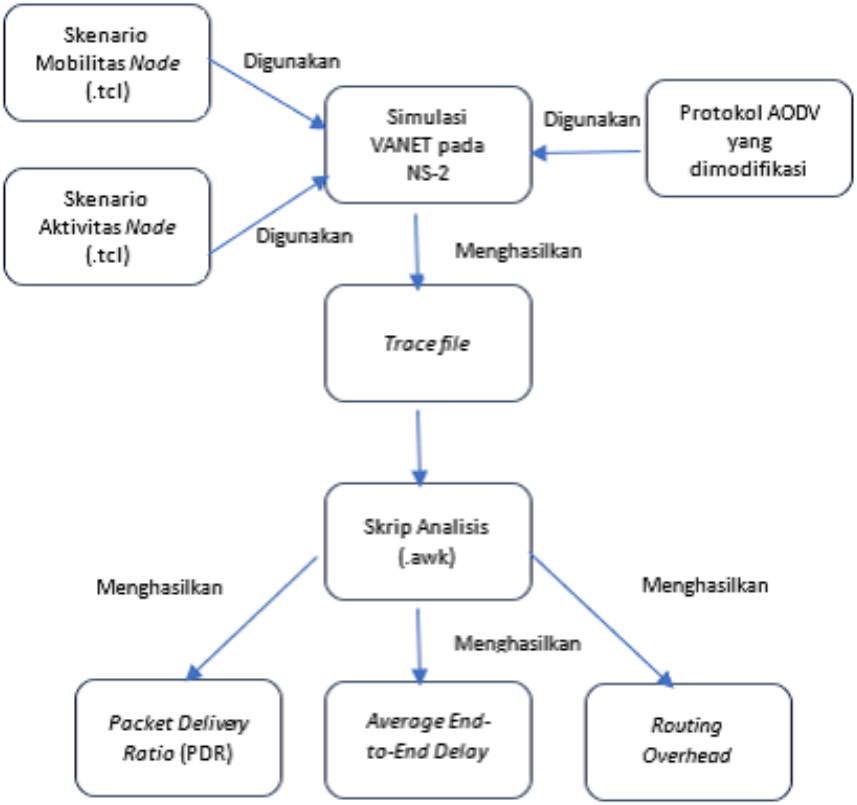
3.1 Deskripsi Umum

Pada Tugas Akhir ini penulis akan mengimplementasikan *routing protocol* AODV yang dimodifikasi dengan menambahkan proses seleksi pemilihan *forwarding node* yang berhak melakukan *rebroadcast* paket *route request* (RREQ) yang akan diteruskan. Modifikasi yang dilakukan yaitu dengan cara menyeleksi *node-node* mana saja yang mempunyai jumlah tetangga dan jumlah *routing table* kurang dari *threshold*. Jika *node* tersebut mempunyai jumlah tetangga dan jumlah *routing table* kurang dari *threshold* maka *node* tersebut berhak meneruskan paket RREQ, jika sebaliknya maka tidak meneruskan paket RREQ atau paket akan di-*drop*.

Modifikasi *routing protocol* AODV ini akan disimulasikan menggunakan NS-2 dengan skenario pergerakan *node* yang dibuat menggunakan *tools* SUMO. Simulasi yang dilakukan menghasilkan *trace file,* yang kemudian dianalisis menggunakan AWK untuk mendapatkan *packet delivery ratio (PDR), average end-to-end delay*, *routing overhead* dan *forwarded route request* (RREQ F). Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV asli. Diagram rancangan simulasi dengan *routing protocol* AODV asli dan AODV yang telah dimodifikasi dapat dilihat pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1 Diagram Rancangan Simulasi dengan *Routing Protocol* AODV Asli



Gambar 3.2 Diagram Rancangan Simulasi dengan *Routing Protocol* AODV yang telah dimodifikasi

Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1	AODV	*Ad hoc On-demand Distance Vector.*
2	PDR	*Packet Delivery Ratio*
3	E2E	*Average End-to-End Delay*

No.	Istilah	Penjelasan
4	RO	*Routing Overhead*
5	RREQ	*Route Request*
6	RREP	*Route Reply*
7	*Threshold*	Batas nilai yang dijadikan sebagai acuan

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas pada Tugas Akhir ini mencakup pada perancangan area peta, pergerakan *node*, dan implementasi pergerakan *node*. Pembuatan peta skenario pada Tugas Akhir ini terbagi menjadi dua, yaitu peta berbentuk *grid* dan peta berbentuk *real*. Peta dengan bentuk *grid* adalah peta dengan jalan berpetak-petak sebagai contoh jalan yang sederhana. Peta *grid* dipilih sebagai simulasi awal karena bentuknya yang lebih stabil dan seimbang. Peta *grid* dapat di*generate* dengan menentukan panjang dan jumlah petak area menggunakan beberapa *tools* yang terdapat pada SUMO. Sedangkan untuk peta *real* digunakan peta asli sebagai sebagai bahan acuan untuk area simulasi. Pada Tugas Akhir ini peta *real* didapatkan dengan bantuan OpenStreetMap dan mengambil salah satu area yang ada di Surabaya.

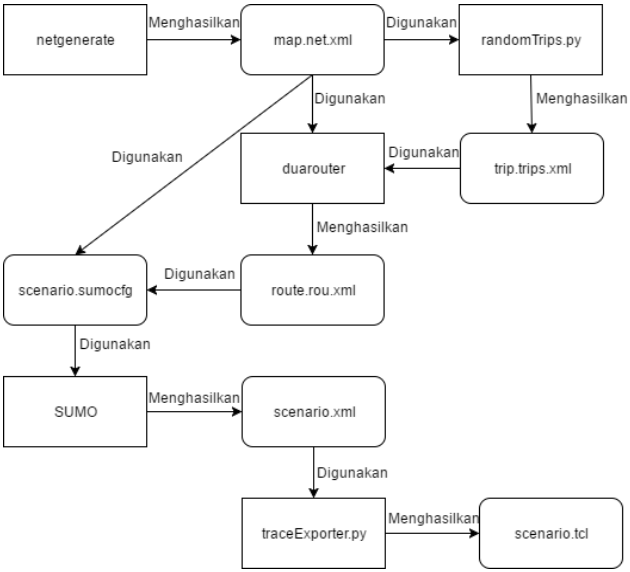
3.2.1 Perancangan Skenario Grid

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan untuk simulasi. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan pada peta *grid*, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak petak yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat digunakan untuk menentukan panjang tiap petak sehingga mendapatkan luas area yang dibutuhkan adalah 1300 m x 1300 m. Dengan 4 titik persimpangan, maka akan didapatkan 9 petak.

Dengan begitu panjang tiap petak untuk mendapatkan luas area 1300 m x 1300 m adalah 400 m.

Peta *grid* yang telah ditentukan luas areanya tersebut kemudian dibuat dengan menggunakan *tools* SUMO yaitu netgenerate. Selain titik persimpangan dan panjang tiap petak *grid*, dibutuhkan juga pengaturan kecepatan kendaraan dalam pembuatan peta grid menggunakan netgenerate ini. Peta *grid* yang dihasilkan oleh netgenerate akan memiliki ekstensi .net.xml. Peta *grid* ini kemudian digunakan untuk membuat pergerakan *node* dengan *tools* SUMO yaitu randomTrips dan duarouter.

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah di*generate*, yang akan menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat menerapkannya pada NS-2 *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan *tools* traceExporter. Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.3.

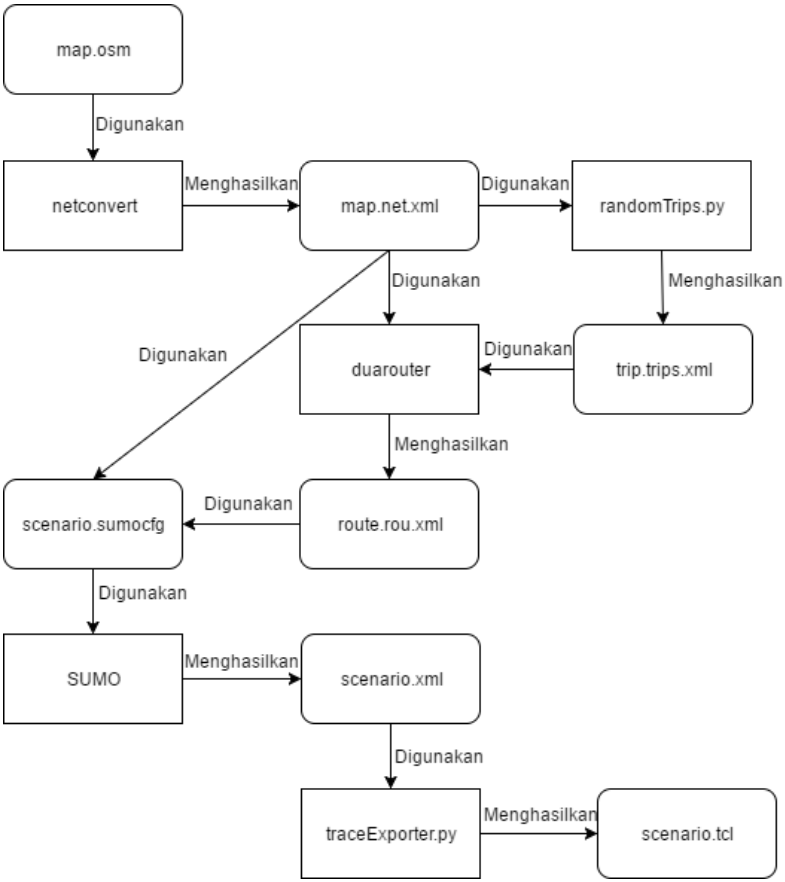


Gambar 3.3 Alur Pembuatan Skenario Mobilitas *Grid*

3.2.2 Perancangan Skenario Real

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, penulis menggunakan peta dari bantuan OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Pada OpenStreetMap bisa memilih area dengan cara mengisi koordinat yang dibutuhkan untuk area yang spesifik ataupun dapat menggunakan *tools* yang telah tersedia di OpenStreetMap dengan cara menggunakan opsi memilih peta secara manual. Setelah memilih area, unduh dengan menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm.

Untuk proses selanjutnya sebenarnya sama seperti merancang skenario mobilitas *grid*, hanya saja terdapat perbedaan penggunaan *tools* untuk mengkonversi peta dari OpenStreetMap tersebut. Setelah mendapatkan peta area yang dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu netconvert. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario mobilitas *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian gabungkan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah di*generate*. Hasil dari penggabungan tersebut merupakan *file* skenario berekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* .tcl agar dapat digunakan untuk simulasi pada NS-2. Alur perancangan skenario mobilitas *real* dapat dilihat pada Gambar 3.4.



Gambar 3.4 Alur Pembuatan Skenario Mobilitas *Real*

3.3 Analisis dan Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme *route discovery* pada protokol tersebut. Pada protokol

AODV, mekanisme pencarian *node* untuk pengiriman ulang *(rebroadcast)* paket RREQ langsung dikirim begitu saja, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*. Seleksi *node* dilakukan dengan cara *node* sumber mengetahui jumlah tetangga yang dimiliki *node* perantara *(one-hop node)* dan jumlah *routing table* yang dimiliki *node* perantara. Selanjutnya, dilakukan perbandingan menggunakan threshold yang sudah ditentukan sebelumnya untuk pengiriman RREQ. Apabila jumlah tetangga dan jumlah *routing table* pada *one-hop node* tersebut lebih kecil atau sama dengan dengan threshold yang sudah ditentukan, maka pengiriman RREQ akan dilanjutkan. Namun sebaliknya, apabila jumlahnya lebih dari threshold, maka paket akan didrop. Jika sudah mencapai node tujuan, node akan mengembalikannya dengan paket RREP. Rute yang dilalui paket RREP adalah rute dengan pembatasan forwarding node yang sudah dilakukan sebelumnya. Rute tersebut tidak melakukan rebroadcast RREQ ke semua node dan paket RREQ hanya melewati node – node terpilih untuk sampai ke node tujuan. Karena beberapa paket RREQ di-drop, maka kemacetan dan tabrakan paket pada jaringan akan lebih rendah sehingga bisa menaikkan PDR.

3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga untuk Setiap *Node*

Pada Tugas Akhir ini, penghitungan jumlah tetangga dilakukan dengan memanfaatkan HELLO *messages* yang ada pada protokol AODV. HELLO *packets* akan mengirimkan HELLO *messages* secara terus menerus untuk memberikan informasi kepada *node* tersebut mengenai tetangga yang ada di sekitarnya. Setiap *node* yang menerima HELLO *messages* dari *node* lainnya, sudah dipastikan menjadi tetangga *node* tersebut. Dengan begitu, jumlah tetangga dapat dihitung dari setiap *node* yang mengirimkan HELLO *messages*. Modifikasi akan dilakukan dengan melakukan penambahan jumlah tetangga pada fungsi nb_insert() dan pengurangan jumlah tetangga pada fungsi nb_delete() pada file aodv.cc yang terletak di dalam folder

ns-2.35/aodv. *Pseudocode* untuk perhitungan jumlah *node* tetangga dapat dilihat pada Gambar 3.5.

```
nb_insert() {
countNeib[index] = countNeib[index] + 1;}
nb_delete() {
countNeib[index] = countNeib[index] - 1;}
```

Gambar 3.5 *Pseudocode* Penghitungan Jumlah *Node*

3.3.2 Perancangan Penghitungan Jumlah *Routing Table* untuk Setiap *Node*

Pada Tugas Akhir ini, Perhitungan jumlah *routing table* dilakukan dengan menggunakan fungsi rt_update() untuk setiap kali penambahan *routing table* dan fungsi rt_down() untuk setiap kali pengurangan jumlah *routing table* pada setiap *node* fungsi rt_update() dan rt_down tersebut terletak pada pada file aodv.cc yang terletak di dalam *folder* ns-2.35/aodv. *Pseudocode* untuk perhitungan jumlah *node* tetangga dapat dilihat pada Gambar 3.6.

```
rt_update() {
countRoute[index] = countRoute[index] + 1;}
rt_down() {
countRoute[index] = countRoute[index] - 1;}
```

Gambar 3.6 *Pseudocode* Penghitungan Jumlah *Routing Table*

3.3.3 *Perancangan Pemilihan Forwarding Node*

Setelah melakukan penghitungan jumlah *node* tetangga, dan jumlah *routing table* akan dilakukan pemilihan *forwarding node*,

yaitu *node* yang berhak untuk melakukan *rebroadcast* paket RREQ. Penentuan *forwarding node* dilakukan dengan cara membandingkan jumlah *node* tetangga dan jumlah *routing table* dengan *threshold* atau batas nilai yang sudah ditentukan. Apabila jumlah tetangga yang dimiliki *node* tersebut kurang dari *threshold* tetangga, dan jumlah *routing table* pada node tersebut kurang dari *threshold routing table* maka *node* tersebut berhak melakukan *rebroadcast*, jika sebaliknya maka paket akan di-*drop*. Langkah tersebut dilakukan untuk mengurangi jumlah paket RREQ yang dikirim sehingga dapat mengurangi kemacetan yang terjadi pada jaringan. *Pseudocode* untuk pemilihan *forwarding node* dapat dilihat pada Gambar 3.7. :

```
if (neighbor <= nbThreshold && rtTable <=
rtThreshold)
then
      receive route request
else
      return
```

Gambar 3.7 *Pseudocode* Pemilihan *Forwarding Node*

3.4 Perancangan Simulasi pada NS-2

Simulasi VANETs pada NS-2 dilakukan dengan menggabungkan *file* skenario yang telah dibuat menggunakan SUMO dan *file* skrip tcl yang berisikan konfigurasi lingkungan simulasi. Konfigurasi lingkungan simulasi VANETs pada NS-2 dapat dilihat pada Tabel 3.2.

Tabel 3.2 Parameter Lingkungan Simulasi dengan Skenario

No.	Parameter	Spesifikasi
1.	*Network Simulator*	NS-2, 2.35
2.	*Routing Protocol*	AODV

3.	Waktu Simulasi	200, 300, 500
4.	Area Simulasi	1300 m x 1300 m dan 1500 m x 1500 m
5.	Banyak Kendaran	60, 150, 300
6.	Agen Pengirim	*Constant Bit Rate* (CBR)
7.	*Protocol* MAC	IEEE 802.11
8.	Propagasi sinyal	*Two-ray ground*
9.	*Source/Destination*	Statis
10.	Tipe Kanal	*Wireless Channel*

Kode yang diubah diantaranya adalah penghitungan jumlah *node* tetangga, jumlah *routing table*, dan perbandingan dengan threshold pada file aodv.cc. Pada saat simulasi NS-2 dijalankan, maka routing protocol AODV akan menyeleksi node mana saja yang berhak melakukan rebroadcast paket RREQ.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter-parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang telah dilakukan modifikasi dan *routing protocol* AODV yang asli:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet Delivery Ratio (PDR) merupakan perbandingan antara jumlah paket yang diterima dengan jumlah paket yang dikirimkan. PDR dihitung dengan persamaan 3.1.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad (3.1)$$

PDR dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi PDR, artinya semakin berhasil pengiriman paket yang dilakukan.

3.5.2 Rata-rata *End-to-End Delay* (E2E)

Rata-rata *End to End Delay* merupakan rata-rata dari *delay* atau waktu yang dibutuhkan tiap paket untuk sampai ke *node* tujuan dalam satuan detik. *Delay* tiap paket didapatkan dari rentang waktu antara *node* asal mengirimkan paket (`CBRSentTime`) dan *node* tujuan menerima paket (`CBRRecvTime`). Dari *delay* tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima (`recvnum`), maka akan didapatkan rata-rata *end to end delay*, yang dapat dihitung dengan persamaan 3.2.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime - CBRSentTime}{recvnum} \qquad (3.2)$$

3.5.3 *Routing Overhead* (RO)

Routing Overhead adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing Overhead* didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route reply* (RREP), maupun *route error* (RERR). Perhitungan *Routing Overhead* dapat dilihat dengan persamaan 3.3.

$$RO = \sum_{m=1}^{sent\ and\ forwarded\ num} packet\ sent \qquad (3.3)$$

3.5.4 *Forwarded Route Request* (RREQ F)

Forwarded Route Request adalah jumlah paket control *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. *Forwarded Route Request* didapatkan dengan menjumlahkan semua paket control *routing* yang ditransmisikan khususnya *route request* bagian *forwarding* (RREQ F). Perhitungan *Forwarded Route Request* dapat dilihat dengan persamaan 3.4

$$Forwarded\ Route\ Request = \sum_{n=1}^{rreqsent} packet\ sent \quad (3.4)$$**s**

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program.

4.1 Implementasi Skenario Mobilitas

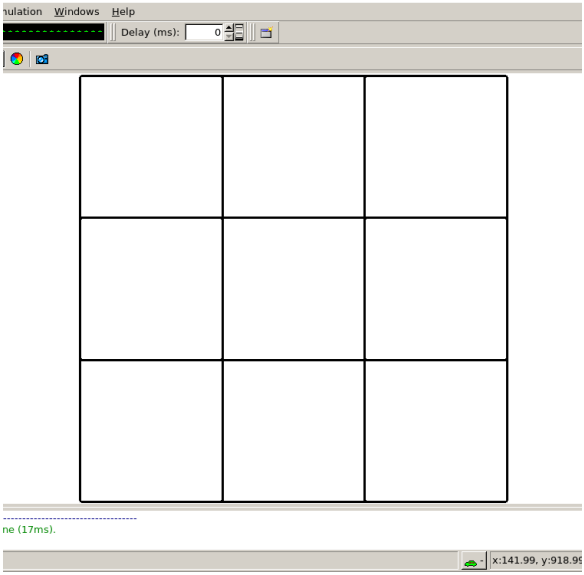
Implementasi skenario mobilitas VANETs dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1300 m x 1300 m yang terdiri dari titik persimpangan antara jalan vertical dan jalan horizontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, maka terbentuk 9 buah petak. Sehingga untuk mencapai luas area sebesar 1300 m x 1300 m dibutuhkan luas per petak sebesar 400 m x 400 m. Perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20 m/s dapat dilihat pada perintah berikut:

```
netgenerate --grid --grid.number=4 --
grid.length=400 --default.speed=20 --
tls.guess=1 --output-file=map.net.xml
```

Setelah itu akan didapat file map berekstensi .xml. Gambar hasil peta yang telah dibuat dengan netgenerate dapat dilihat pada Gambar 4.1.



Gambar 4.1 Hasil Generate Peta Grid

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik awal dan titik akhir setiap *node* seacara random menggunakan *tools* randomTrips yang terdapat di SUMO. Perintah penggunaan *tools* randomTrips untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.2.

```
python $SUMO_HOME/tools/randomTrips.py -n
map.net.xml -e 58 -l --trip-
attributes="departLane=\"best\"
departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.2 Perintah randomTrips

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools* duarouter. Secara *default* algoritma yang digunakan untuk membuat rute ini adalah algoritma djikstra. Penggunaan *tools* duarouter dapat dilihat pada perintah berikut:

```
duarouter -n map.net.xml -t trip.trips.xml -
o route.rou.xml --ignore-errors --repair
```

Ketika menggunakan *tools* duarouter, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tools* randomTrips. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg untuk menggabungkan *file* peta dan rute pergerakan *node*. Isi dari *file* skrip .sumocfg dapat dilihat pada Gambar 4.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
xsi:noNamespaceSchemaLocation="http://sumo.
dlr.de/xsd/sumoConfiguration.xsd">
   <input>
      <net-file value="map.net.xml"/>
      <route-files value="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="200"/>
   </time>
</configuration>
```

Gambar 4.3 *File* Skrip .sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta dan *file* rute pergerakan *node.* Untuk percobaan sebelum dikonversi, *file* .sumocfg dapat dibuka dengan menggunakan *tools* sumo-gui. Kemudian buat *file* skenario dalam bentuk *file* .xml dari sebuah *file* skrip berekstensi .sumocfg menggunakan *tools* SUMO. Untuk menggunakan *tools* SUMO dapat dilihat pada perintah berikut:

```
sumo -c file.sumocfg --fcd-output
scenario.xml
```

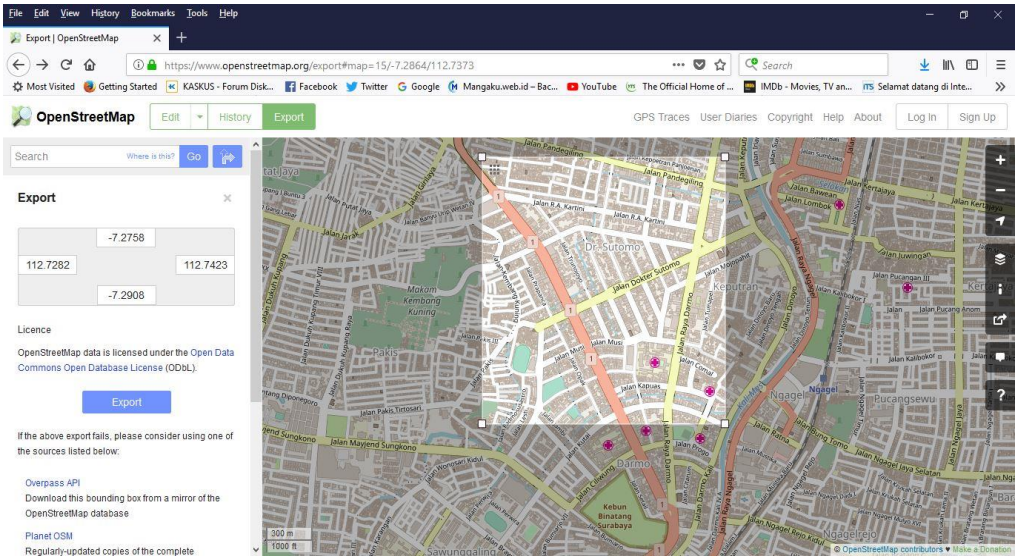
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.4.

```
python $SUMO_HOME/tools/traceExporter.py --
fcd-input=scenario.xml --ns2mobility-
output=scenario.tcl
```

Gambar 4.4 Perintah traceExporter

4.1.2 Skenario *Real*

Dalam mengimplementasikan skenario *real*, langkah pertama adalah menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.5.

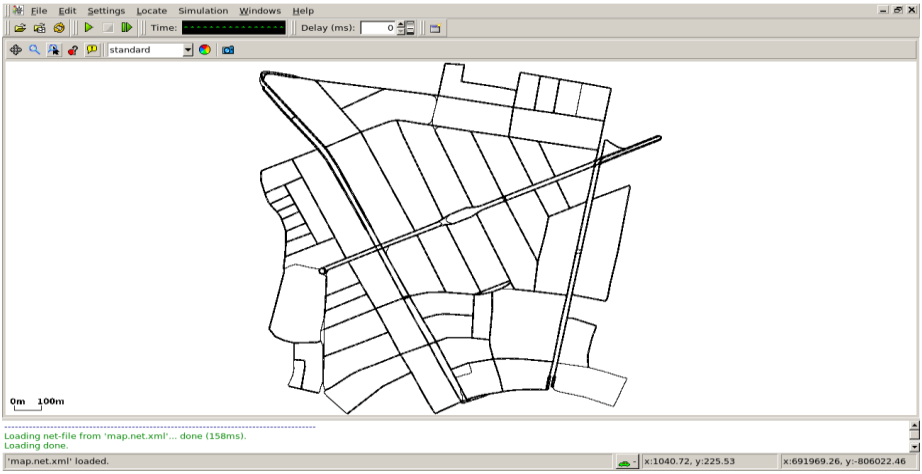


Gambar 4.5 Ekspor Peta dari OpenStreetMap

File hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada perintah berikut:

```
netconvert --osm-files map.osm --output-
file map.net.xml
```

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.6.



Gambar 4.6 Hasil Konversi Peta *Real*

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tools* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tools* duarouter. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tools* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya konversikan *file* skenario berekstensi .tcl untuk dapat disimulasikan pada NS-2 menggunakan *tools* traceExporter. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Memilih *Forwarding Node*

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu node yang bertugas untuk melakukan rebroadcast paket RREQ.

Hal tersebut dilakukan dengan cara memilih *forwarding node* berdasarkan jumlah *node* tetangga dan jumlah *routing table* dari node tersebut dengan threshold yang sudah ditentukan, sehingga dapat dilihat peningkatan performa pada routing AODV yang telah dimodifikasi. Implementasi modifikasi routing protocol AODV ini dibagi menjadi 3 bagian yaitu:

- Implementasi Penghitungan Jumlah *Node* Tetangga
- Implementasi Penghitungan Jumlah *Routing Table*
- Implementasi Pemilihan *Forwarding Node*

Kode implementasi dari routing protocol AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Pada direktori tersebut terdapat beberapa file diantaranya seperti aodv.cc, aodv.h dan sebagainya. Pada Tugas Akhir ini, penulis memodifikasi file aodv.cc yang ada di dalam folder ns-2.35/aodv untuk menghitung jumlah *node* tetangga, menghitung jumlah *routing table*, dan menentukan *forwarding node*. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi routing protocol AODV untuk mengurangi jumlah *forwarding node* yang melakukan *rebroadcast.*

4.2.1 Implementasi Menghitung Jumlah *Node* Tetangga

Langkah awal yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah dengan cara menangkap pesan HELLO messages dari node yang mengirimkannya. Node yang mengirimkan HELLO messages sudah dapat dipastikan berada di sekitar node tersebut dan berada di range transmisi dari node yang sedang dieksekusi. Secara default, AODV menginformasikan siapa saja node tetangga yang ada di sekitar node tersebut. dan apabila terdapat node tetangga menjauh kita harus mengurangi jumlah tetangga dari node tersebut Tugas Akhir ini memanfaatkan fungsi nb_insert() dan nb_delete() pada kode sumber

aodv.cc yang terdapat dalam folder ns2.35/aodv. Untuk potongan kode tersebut bisa dilihat pada Gambar 4.7

```
void
AODV::nb_insert(nsaddr_t id) {
countNeib[index] = countNeib[index] + 1;}
void
AODV::nb_delete(nsaddr_t id) {
countNeib[index] = countNeib[index] - 1 ;}
```

Gambar 4.7 Potongan Kode untuk menghitung jumlah node tetangga

4.2.2 Implementasi Menghitung Jumlah *Routing Table*

Langkah awal yang dilakukan untuk menghitung jumlah routing table adalah menghitung setiap node yang berhasil diinputkan pada routing table kemudian kita mengurangkan dengan node yang dihapus dari *routing table*. Tugas Akhir ini memanfaatkan fungsi rt_update() dan rt_down() pada kode sumber Gambar 4.8

```
void AODV::rt_update(aodv_rt_entry *rt, u_int32_t seqnum,
u_int16_t metric, nsaddr_t nexthop, double expire_time) {
countRoute[index] = countRoute[index] + 1;
}
void AODV::rt_down(aodv_rt_entry *rt) {
countRoute[index] = countRoute[index] - 1;}
```

Gambar 4.8 Potongan Kode untuk Menghitung jumlah *Routing Table*

4.2.3 Implementasi Pemilihan *Forwarding Node*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga dan jumlah *routing table*, langkah yang harus dilakukan selanjutnya adalah pemilihan *forwarding node* untuk melanjutkan paket RREQ Penghitungan ini dilakukan pada fungsi recvRequest() yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Jumlah tetangga yang sudah ditemukan, disimpan dalam variabel countNeib dan Jumlah *routing table* disimpan pada countRoute. Jumlah tetangga dan *routing table* kemudian akan dibandingkan dengan threshold yang sudah ditentukan dan bukan merupakan node sumber. Nilai threshold didapatkan dari percobaan dari angka 1 sampai batas maksimal node yang dilakukan selama 10 kali dan ditentukan dari nilai yang paling sering muncul dalam percobaan tersebut. Apabila jumlah tetangga lebih dari *threshold* tetangga dan jumlah *routing table* lebih dari threshold *routing table* yang ditentukan dan bukan merupakan node sumber, maka paket RREQ akan didrop. Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.9.

```
if (countNeib[index] > NeibThreshold &&
countRoute[index] > RouteThreshold && rq-
>rq_dst != index){
Packet::free(p);
return;
}
```

Gambar 4.9 Potongan Kode Untuk Penyeleksian Forwarding Node

Dengan proses penyeleksian *node* mana saja yang dapat menerima paket RREQ sudah dapat mengurangi jumlah paket *routing overhead* dan mengurangi jumlah *forwarded route request*. Dengan begitu akan mengurangi *cost* untuk pengiriman paket.

4.3 Implementasi Simulasi pada NS-2 dan *Traffic* Pengiriman Dua Sesi

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* berekstensi .tcl. *File* ini berisikan konfigurasi yang dibutuhkan untuk setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.10.

```
set val(chan)    Channel/WirelessChannel
set val(prop)    Propagation/TwoRayGround
set val(netif)   Phy/WirelessPhy
set val(mac)     Mac/802_11
set val(ifq)     Queue/DropTail/PriQueue
set val(ll)      LL
set val(ant)     Antenna/OmniAntenna
set opt(x)       1500
set opt(y)       1500
set val(ifqlen)  1000
set val(nn)      60
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr1.txt"
set val(sc)      "scen1modif.txt"
```

Gambar 4.10 Konfigurasi Lingkungan Simulasi

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, dan pengiriman paket dengan dua sesi, serta *file* skenario yang berisi pergerakan *node* yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.6 Kode Skenario NS-2

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan keluar sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *plain text* berekstensi .tr. Dari data *trace file* tersebut, dapat dilakukan analisis performa *routing protocol* dengan mengukur beberapa metrik. Pada Tugas Akhir ini, metrik yang akan dianalisis adalah *Packet Delivery Ratio* (PDR), Rata-rata *End-to-End Delay* (E2E), dan *Routing Overhead* (RO).

4.4.1 Implementasi *Packet Delivery Ratio*

Pada subbab 2.3.2 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan 3.1 telah dijelaskan bagaimana menghitung PDR. Dengan begitu, melakukan perhitungan PDR melalui bantuan skrip awk. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut dapat dilihat pada lampiran A.8 Kode Skrip AWK *Packet Delivery Ratio*.Dalam perhitungan PDR, kata kunci yang perlu diperhatikan dari *trace file* adalah layer AGT, karena kata kunci tersebut menunjukkan *event* yang bersangkutan dengan paket komunikasi data. Kemudian hitung jumlah paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*, karena kolom pertama yang menunjukkan *event* yang terjadi dari sebuah paket. Setelah itu nilai PDR dapat dihitung dengan persamaan yang telah dijelaskan. *Pseudocode* untuk menghitung PDR dapat dilihat pada Gambar 4.11.

```
sent = 0
received = 0
for i = 1 to the number of rows
     if in a row contains "s" and AGT then
          sent++
     else if in a row contains "r" and AGT then
          received++
     end if
pdr = received / sent
```

Gambar 4.11 *Pseudocode* untuk Menghitung PDR

Perhitungan PDR pada Tugas Akhir ini dimulai dari sesi kedua karena implementasi Tugas Akhir ini diimplementasikan pada sesi kedua. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f pdr.awk scenario1.tr`.

4.4.2 Implementasi Rata-Rata *End-to-End Delay*

Perhitungan E2E telah dijelaskan melalui persamaan. Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.9 Kode Skrip AWK Rata-Rata *End-to-End Delay*.

Dalam perhitungan E2E, langkah yang digunakan untuk mendapatkan E2E hampir sama dengan ketika mencari PDR, hanya saja yang perlu diperhatikan adalah waktu dari sebuah *event* yang tercatat pada kolom ke-2 dengan *filter event* pada kolom ke-4 adalah layer AGT dan *event* pada kolom pertama guna membedakan paket dikirim atau diterima. Setelah seluruh baris yang memenuhi didapatkan, akan dihitung *delay* dari paket dengan mengurangi waktu dari paket diterima dengan waktu dari paket dikirim dengan sayarat memiliki *id* paket yang sama.

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. *Pseudocode* untuk menghitung rata-rata *end-to-end delay* dapat dilihat pada Gambar 4.12.

```
sum_delay = 0
counter = 0
for i = 1 to the number of rows
     counter++
     if layer == AGT and event == s then
      start_time[packet_id] = time
     else if layer == AGT and event == r then
      end_time[packet_id] = time
     end if
     delay[packet_id] = end_time[packet_id] –
start_time[packet_id]
     sum_delay += delay[packet_id]
e2e = sum_delay / counter
```

Gambar 4.12 *Pseudocode* untuk Perhitungan Rata-Rata *End-to-End Delay*

Pada Tugas Akhir ini, perhitungan *end-to-end delay* juga dilakukan saat sesi kedua seperti perhitungan *packet delivery ratio*. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk –f e2e.awk sceanrio1.tr`.

4.4.3 Implementasi *Routing Overhead*

Perhitungan *routing overhead* telah dijelaskan pada persamaan 3.3. Skrip awk untuk menghitung *routing overhead* dapat dilihat pada lampiran A.10 Kode Skrip AWK *Routing Overhead*.

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu paket RREQ, paket RREP, maupun paket RERR. Dengan begitu, untuk mendapatkan *route request* yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. *Pseudocode* untuk menghitung *routing overhead* dapat dilihat pada Gambar 4.13.

```
ro = 0
for i = 1 to the number of rows
     if in a row contains "s" or "f" and RTR
then
        ro++
     end if
```

Gambar 4.13 *Pseudocode* untuk Perhitungan *Routing Overhead*

Pada Tugas Akhir ini, perhitungan *routing overhead* juga dilakukan pada sesi kedua seperti perhitungan *packet delivery ratio* dan *end-to-end delay*. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f ro.awk scenario1.tr`.

4.4.4 Implementasi *Forwarded Route Request*

Perhitungan *Forwarded Route Request* telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung *forwarded route request* dapat dilihat pada lampiran A.11 Kode Skrip AWK *Forwarded Route Request*.

Seperti yang telah dijelaskan sebelumnya, *forwarded route request* adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. Dengan begitu, untuk mendapatkan *forwarded route request* yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event forwarded* pada kolom pertama, *event layer* RTR pada kolom ke-4 dan *event layer*

route request pada kolom-19. *Pseudocode* untuk menghitung *forwarded route request* dapat dilihat pada Gambar 4.14.

```
rreqf = 0
for i = 1 to the number of rows
     if in a row contains "f" and RTR and route
request then
        rreqf++
     end if
```

Gambar 4.14 *Pseudocode* untuk Perhitungan *Forwarded Route Request*

Pada Tugas Akhir ini, perhitungan *forwarded route request* juga dilakukan pada sesi kedua seperti perhitungan *packet delivery ratio*, *end-to-end delay*, dan *routing overhead*. Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f rreqf.awk scenario1.tr.`

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan tahap uji coba dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat ditarik kesimpulan pada bab selanjutnya.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

Komponen	**Spesifikasi**
CPU	AMD A10-5750M QUAD 2.5-3.5GHz
Sistem Operasi	Ubuntu 14.04.5 LTS (Trusty Tahr)
Linux Kernel	Linux kernel 4.4
Memori	4.0 GB
Penyimpanan	40 GB

Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* menggunakan kode pada masing-masing lampiran A.8 Kode Skrip AWK *Packet Delivery Ratio*, A.9 Kode Skrip AWK Rata-Rata *End-to-End Delay*, A.10 Kode Skrip AWK *Routing Overhead*, dan A.11 Kode Skrip AWK *Forwarded Route Request*.

5.2 Hasil Uji Coba

Untuk hasil uji coba dari skenario *grid* dan skenario *real* untuk Tugas Akhir ini dapat dilihat sebagai berikut:

5.2.1 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* antara *routing protocol* AODV asli dan *routing protocol* AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*. Pengujian dilakukan sesuai dengan perancangan *parameter* lingkungan simulasi yang dapat dilihat pada Tabel 3.2.

Pengambilan data *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* dilakukan dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 untuk lingkungan yang sedang, dan 300 untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. dan dilakukan uji coba dari 1 hingga 5 koneksi. Untuk uji coba setiap lingkungan menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga dan jumlah *routing table* yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut.

Hasil pengambilan data pada skenario grid 60 node 1 koneksi hingga 5 koneksi dengan menggunakan AODV asli dan AODV yang telah dimodifikasi kemudian di rata rata lagi lalu dibandingkan antara lingkungan dengan node 60, 150 dan 300 dari 1 koneksi hingga 5 koneksi dengan menggunakan AODV murni dan AODV yang telah dimodifikasi dapat dilihat pada Tabel 5.2, Tabel 5.3, Tabel 5.4, dan Tabel 5.5

Tabel 5.2 Hasil Perhitungan Rata - Rata PDR pada Skenario Grid

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	84.85%	85.44%	0.59%
150 Node	92.72%	90.57%	2.14%
300 Node	92.02%	87.17%	4.84%

Tabel 5.3 Hasil Perhitungan Rata - Rata *Routing Overhead* pada Skenario *Grid*

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	18773.08	20112	1338.92
150 Node	73694.7	84185.26	10490.56
300 Node	63916.36	71452.78	7536.42

Tabel 5.4 Hasil Perhitungan Rata - Rata *End to End Delay* pada Skenario *Grid*

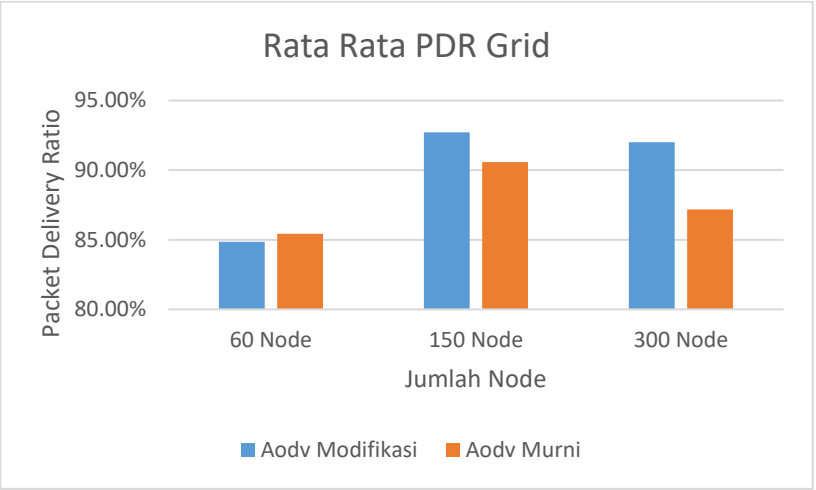
Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	443.7796 ms	394.4155 ms	49.36405 ms
150 Node	320.56512 ms	393.6312 ms	73.06608 ms
300 Node	510.35523 ms	1451.523 ms	941.16838 ms

Tabel 5.5 Hasil Perhitungan Rata-Rata Forwarded Route Request pada Skenario Grid

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	4684.62	5859.74	1175.12
150 Node	10482.66	20706.3	10223.64
300 Node	2900.46	10119.92	7219.46

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan *packet delivery ratio*, *end-to-end delay*, *routing*

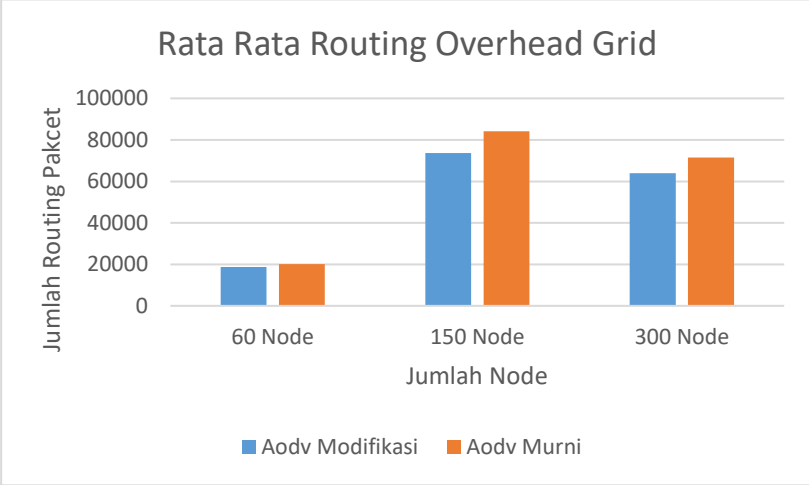
overhead, dan *forwarded route request* yang ditunjukkan pada Gambar 5.1, Gambar 5.2, Gambar 5.3, Gambar 5.4.



Gambar 5.1 Grafik PDR Skenario Grid

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* AODV murni dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif pada *packet delivery ratio*. Pada lingkungan yang jarang dengan *node* berjumlah 60, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 0.59%, dimana terjadi penurunan sebesar 0.59% dan *routing protocol* AODV yang murni unggul dalam hal *packet delivery ratio* tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 2.14%, dimana terjadi kenaikan *packet delivery ratio* sebesar 2,14% dan *routing protocol* AODV yang dimodifikasi unggul dalam hal *packet delivery ratio* tersebut dari AODV murni. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 4.84%, dimana terjadi kenaikan *packet delivery ratio* sebesar 4.84% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *packet delivery ratio* tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang banyak atau pada lingkungan yang padat, AODV yang di modifikasi menghasilkan *packet delivery ratio* yang lebih baik daripada di lingkungan dengan jumlah *node* yang jarang maupun sedang apabila dibandingkan dengan *routing protocol* AODV murni. Tetapi apabila kita lihat selisih kenaikan *packet delivery ratio* pada setiap skenario nya kita bisa melihat bahwa selisih kenaikan *packet delivery ratio* antara AODV dan AODV yang telah dimodifikasi selalu meningkat dengan semakin padatnya jumlah node, hal ini disebabkan karena semakin padat node pada suatu skenario semakin banyak juga node yang bisa menjadi rute pengganti saat jumlah *routing table* pada suatu rute menjadi padat, sehingga membuat selisih peningkatan *packet delivery ratio* semakin tinggi. Nilai rata-rata kenaikan *packet delivery ratio* pada skenario grid adalah sebesar 2.13%. Untuk hasil pengambilan data *routing overhead* pada skenario grid 60 node, 150 node, dan 300 node dapat dilihat pada Gambar 5.2.



Gambar 5.2 Grafik Routing Overhead Skenario Grid

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa routing protocol AODV yang telah dimodifikasi dan juga routing protocol AODV asli mengalami kenaikan yang fluktuatif pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 node, menghasilkan perbedaan selisih *routing overhead* sebesar 1338.92, dimana terjadi penurunan *routing overhead* sebesar 6.65% dan routing protocol AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan RO yang lebih rendah dari AODV murni.

Pada lingkungan yang sedang dengan jumlah 150 node, menghasilkan perbedaan selisih routing overhead sebesar 10490.56, dimana terjadi penurunan *routing overhead* sebesar 12.46% dan routing protocol AODV yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut dari AODV murni.

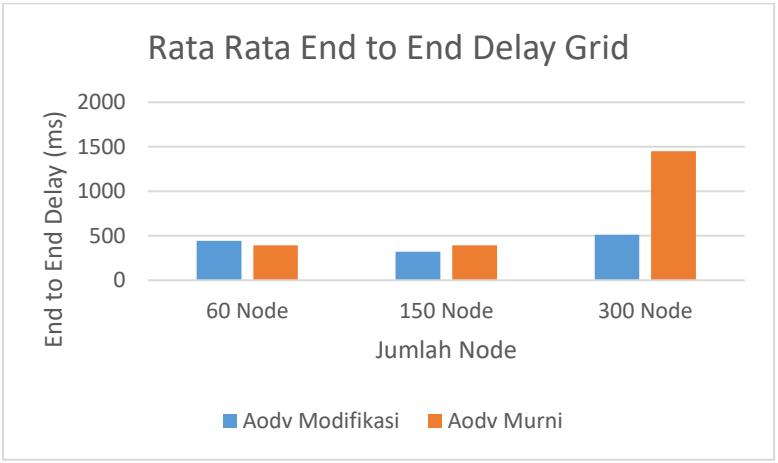
Pada lingkungan yang padat dengan jumlah 300 node, menghasilkan perbedaan selisih RO sebesar 7536.42, dimana terjadi penurunan *routing overhead* sebesar 10.54% dan routing protocol AODV yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah node yang lebih sedikit atau pada lingkungan dengan node yang jarang, menghasilkan *routing overhead* yang lebih sedikit daripada di lingkungan dengan jumlah node yang sedang maupun yang padat baik untuk routing protocol AODV asli maupun routing protocol AODV yang telah dimodifikasi.

Terjadi penurunan yang sangat derastis pada skenario dengan menggunakan 300 *node* hal ini disebabkan karena pada saat jumlah node padat di skenario grid, persebaran node sudah merata sehingga tidak diperlukan *forwarding request* berkali kali pada saat pengiriman RREQ dari *source* menuju *destination* yang akan menyebabkan penurunan *routing overhead.*

Nilai rata – rata penurunan *routing overhead* pada AODV yang dimodifikasi adalah sebesar 11.01%. Dapat dilihat pula bahwa

AODV yang telah dimodifikasi menghasilkan *routing overhead* yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.



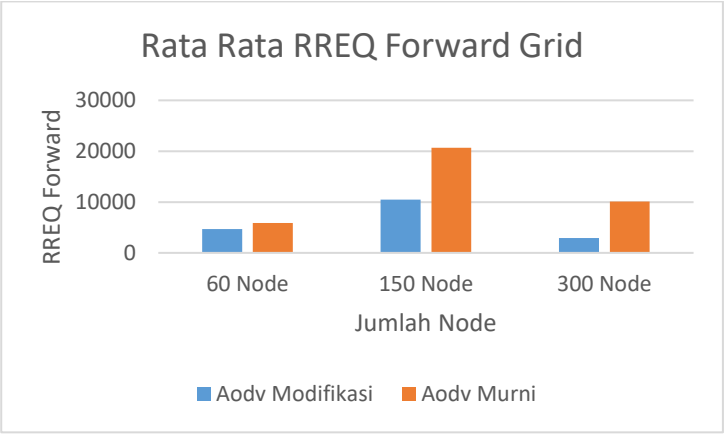
Gambar 5.3 Grafik End to End Delay Skenario Grid

Untuk hasil pengambilan data *End to End Delay* pada skenario grid 60 node, 150 node, dan 300 node dapat dilihat pada Gambar 5.3 Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa rata-rata *end-to-end delay* antara routing protocol AODV asli dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 node, terjadi perbedaan selisih *end-to-end delay* sebesar 49.36 ms dimana terjadi kenaikan sebesar 12.51% antara routing protocol AODV murni dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV murni unggul dalam hal *end-to-end delay* tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 node, terjadi perbedaan selisih *end-to-end delay* sebesar 73.06 ms dimana terjadi penurunan sebesar 18.56% antara routing protocol AODV asli dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV yang telah dimodifikasi lebih unggul dalam hal *end-to-end delay*.

Pada lingkungan yang padat dengan jumlah 300 node, terjadi perbedaan selisih *end-to-end delay* sebesar 941.16 ms antara routing

protocol AODV asli dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika ketiga lingkungan tersebut dibandingkan, semakin padat lingkungan tersebut semakin baik hasil *end-to-end delay* dari AODV modifikasi. Begitu pula sebaliknya. Semakin jarang lingkungan semakin buruk hasil *end-to-end delay* pada AODV yang telah di modifikasi. Routing protocol AODV yang telah dimodifikasi selalu lebih unggul daripada routing protocol AODV yang telah dimodifikasi pada lingkungan sedang dan padat. Tetapi AODV yang telah dimodifikasi kalah pada lingkungan yang jarang. Tetapi apabila kita lihat selisih kenaikan *end-to-end delay* pada setiap skenario nya kita bisa melihat bahwa selisih perbedaan *end-to-end delay* antara AODV dan AODV yang telah dimodifikasi selalu meningkat dengan semakin padatnya jumlah node, hal ini disebabkan karena semakin padat node pada suatu skenario semakin banyak juga node yang bisa menjadi rute pengganti saat jumlah *routing table* pada suatu rute menjadi padat, sehingga membuat selisih peningkatan *end-to-end delay* semakin tinggi. Hasil pengambilan data rata-rata untuk - *forwarded route request* pada skenario *real* dengan jumlah *node* 60, 150, dan 300 dapat dilihat pada Gambar 5.4.



Gambar 5.4 Grafik Forwarded Route Request Skenario Grid

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1175.12, dimana terjadi penurunan *forwarded route request* sebesar 20.05% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV murni. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 10223.64, dimana terjadi penurunan *forwarded route request* sebesar 49.37% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam *forwarded route request* tersebut dari *forwarded route request* AODV murni. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 7219.46, dimana terjadi penurunan *forwarded route request* sebesar 71.33% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut. Terjadi penurunan yang sangat derastis pada skenario dengan menggunakan 300 node hal ini disebabkan karena pada saat jumlah node padat di skenario grid, persebaran node sudah merata sehingga tidak diperlukan *forwarding request* berkali kali pada saat pengiriman RREQ dari *source* menuju *destination*

Jika ketiga lingkungan tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* yang sedikit atau lingkungan jarang menghasilkan *forwarded route request* yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah node yang sedang maupun yang padat baik untuk routing protocol AODV asli maupun routing protocol AODV yang telah dimodifikasi. Nilai rata – rata penurunan yang terjadi pada *forwarded route request* adalah sebesar 50.75%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam

hal ini lebih rendah daripada *forwarded route request* asli dengan jumlah selisih *forwarded route request* yang cukup signifikan.

5.2.2 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan *packet delivery ratio*, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* antara *routing protocol* AODV asli dan *routing protocol* AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*. Pengujian dilakukan sesuai dengan perancangan *parameter* lingkungan simulasi yang dapat dilihat pada Tabel 3.2.

Pengambilan data uji PDR, rata-rata *end-to-end delay*, *routing overhead*, dan *forwarded route request* dengan luas area 1500 m x 1500 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 untuk lingkungan yang sedang, dan 300 untuk lingkungan yang padat. lalu dilakukan uji coba dari 1 hingga 5 koneksi.

Seperti pada uji coba skenario *grid* untuk uji coba setiap lingkungan menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut.

Hasil pengambilan data skenario *real* kemudian di rata rata lagi lalu dibandingkan antara lingkungan dengan node 60, 150 dan 300 dari 1 koneksi hingga 5 koneksi dengan menggunakan AODV murni dan AODV yang telah dimodifikasi dapat dilihat pada Tabel 5.6, Tabel 5.7,Tabel 5.8, dan Tabel 5.9

Tabel 5.6 Tabel Hasil Rata –Rata Perhitungan PDR Skenario Real

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	89.30%	88.76%	0.54%
150 Node	89.04%	88.73%	0.30%

300 Node	89.84%	81.63%	8.21%

Tabel 5.7 Tabel Hasil Rata –Rata Perhitungan Routing Overhead Skenario Real

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	17655.64	19519.68	1864.04
150 Node	39379.48	42658.06	3278.58
300 Node	62341.58	66389.44	4047.86

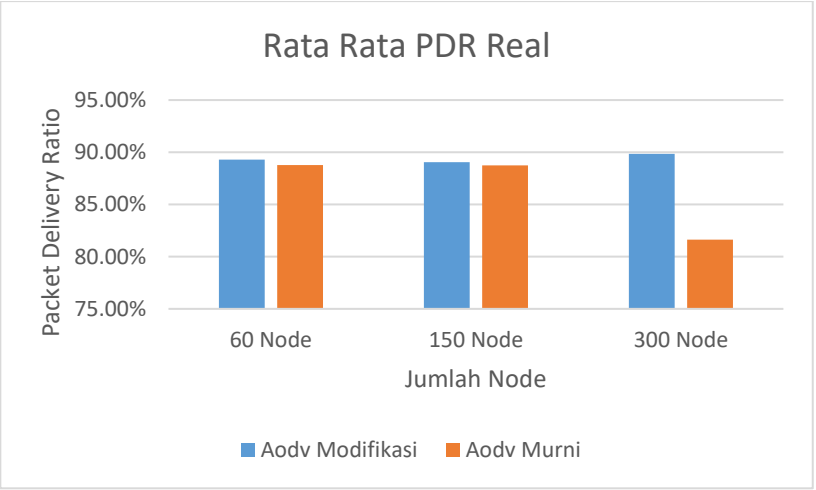
Tabel 5.8 Tabel Hasil Rata –Rata Perhitungan End to End Delay

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	446.288 ms	379.8150 ms	66.4734 ms
150 Node	672.9792 ms	718.3574 ms	45.3781 ms
300 Node	304.3750 ms	1148.865 ms	844.4900 ms

Tabel 5.9 Tabel Hasil Rata –Rata Perhitungan Forwarded Route Request Skenario Real

Jumlah Node	Aodv Modifikasi	Aodv Murni	Perbedaan
60 Node	3573.24	5264.36	1691.12
150 Node	1059.26	4177.84	3118.58
300 Node	1416.556667	5239.66	3823.103333

Dari data di atas, kemudian dibuat lah grafik yang merepresentasikan hasil perhitungan *Packet Delivery Ratio*, *End to End Delay*, *Routing Overhead*, dan *Forwarded Route Request* yang ditunjukkan pada Gambar 5.5,Gambar 5.6,Gambar 5.7,dan Gambar 5.8



Gambar 5.5 Grafik PDR Skenario Real

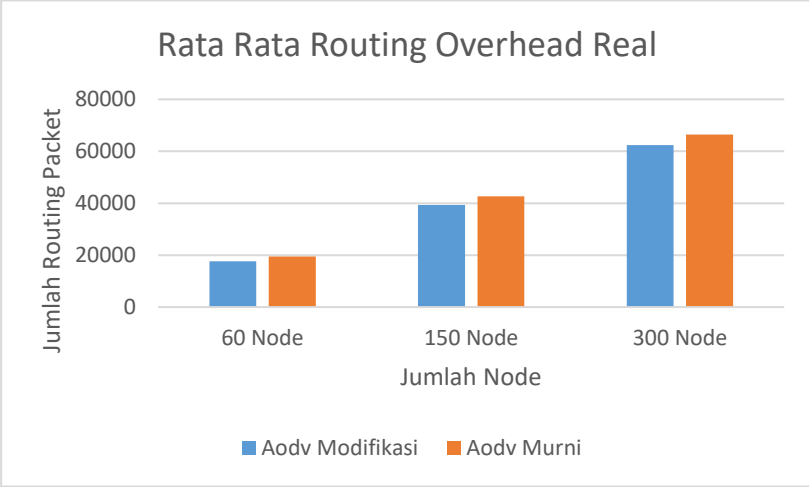
Berdasarkan grafik pada Gambar 5.5, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami perubahan yang tidak signifikan pada *packet delivery ratio*. Pada lingkungan yang jarang dengan *node* berjumlah 60, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 0,54%, dimana terjadi kenaikan sebesar 0.54% dan *routing protocol* AODV yang asli unggul dalam hal *packet delivery ratio* tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 0,30%, dimana terjadi kenaikan *packet delivery ratio* sebesar 0.30% dan *routing protocol* AODV yang dimodifikasi unggul dalam hal *packet delivery ratio* tersebut dari AODV asli.

Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *packet delivery ratio* sebesar 8,2%, dimana terjadi kenaikan *packet delivery ratio* sebesar 8,2% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *packet delivery ratio* tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang banyak atau pada lingkungan yang padat, menghasilkan *packet delivery ratio* yang lebih baik daripada di lingkungan dengan jumlah *node* yang jarang maupun sedang baik untuk *routing protocol* AODV murni maupun *routing protocol* AODV yang telah dimodifikasi.

Tetapi apabila kita lihat selisih kenaikan *packet delivery ratio* pada setiap skenario nya kita bisa melihat bahwa selisih kenaikan *packet delivery ratio* antara AODV dan AODV yang telah dimodifikasi selalu meningkat dengan semakin padatnya jumlah node walaupun tidak signifikan, hal ini disebabkan karena semakin padat node pada suatu skenario semakin banyak juga node yang bisa menjadi rute pengganti saat jumlah *routing table* pada suatu rutes menjadi padat, sehingga membuat selisih peningkatan *packet delivery ratio* semakin tinggi.

Nilai rata-rata kenaikan *packet delivery ratio* pada skenario Real adalah sebesar 3.49%. Untuk hasil pengambilan data *routing overhead* pada skenario *real* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.6



Gambar 5.6 Grafik Routing Overhead Skenario Real

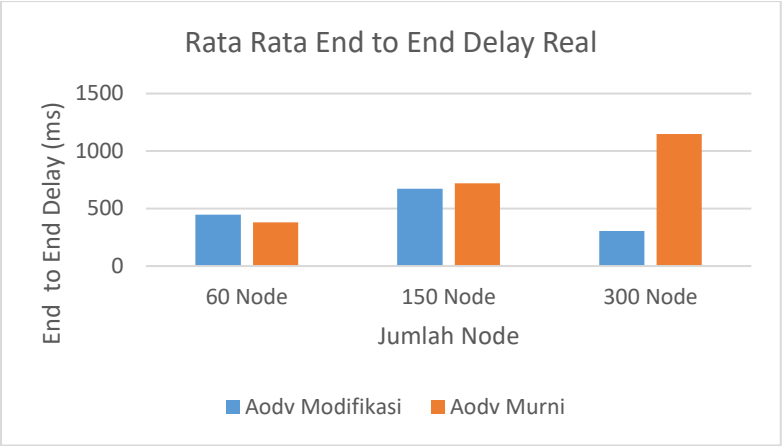
Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa routing protocol AODV yang telah dimodifikasi dan juga routing protocol AODV murni mengalami kenaikan yang signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 node, menghasilkan perbedaan selisih *routing overhead* sebesar 1864.04, dimana terjadi penurunan *routing overhead* sebesar 9.54% dan routing protocol AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut karena menghasilkan *routing overhead* yang lebih rendah dari AODV murni.

Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 3278.58, dimana terjadi penurunan *routing overhead* sebesar 7,68% dan routing protocol AODV yang telah dimodifikasi juga unggul dalam hal *routing overhead* tersebut dari AODV murni.

Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 4047.86, dimana terjadi penurunan *routing overhead* sebesar 6.09% dan routing protocol AODV yang telah dimodifikasi unggul dalam hal *routing overhead* tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah node yang lebih sedikit atau pada lingkungan dengan node yang jarang, menghasilkan *routing overhead* yang lebih sedikit daripada di lingkungan dengan jumlah node yang sedang maupun yang padat baik untuk routing protocol AODV murni maupun routing protocol AODV yang telah dimodifikasi.

Hal ini disebabkan karena semakin banyaknya node maka semakin banyak pula jumlah routing packet yang terjadi. Nilai rata – rata penurunan *routing overhead* pada AODV yang dimodifikasi adalah sebesar 7.14%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV murni.



Gambar 5.7 Grafik End to End Delay Skenario Real

Untuk hasil pengambilan data *end-to-end delay* pada skenario *real* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.7. Berdasarkan grafik pada Gambar 5.7, dapat dilihat bahwa rata-rata end-to-end delay antara routing protocol AODV murni dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif.

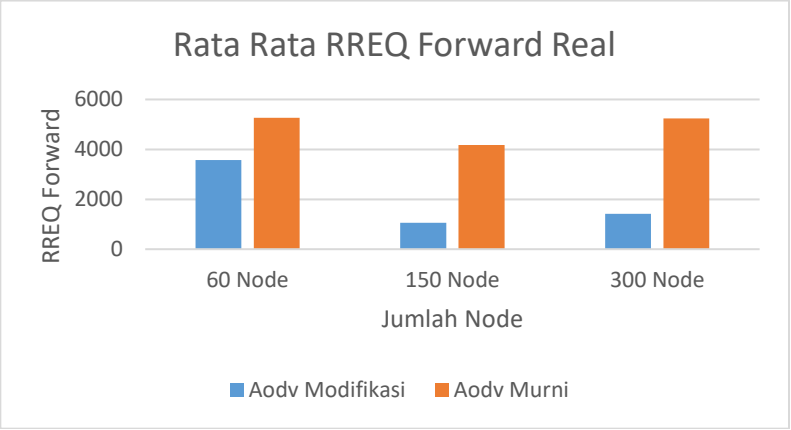
Pada lingkungan yang jarang dengan jumlah 60 node, terjadi perbedaan selisih *end-to-end delay* sebesar 66.47 ms atau terjadi kenaikan *end-to-end delay* sebesar 17.50% antara routing protocol AODV murni dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV murni unggul dalam hal *end-to-end delay* tersebut.

Sedangkan pada lingkungan sedang dengan jumlah 150 node, terjadi perbedaan selisih *end-to-end delay* sebesar 45.37 ms atau terjadi penurunan *end-to-end delay* sebesar 6.31% antara routing protocol AODV asli dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV yang telah dimodifikasi lebih unggul dalam hal *end-to-end delay*.

Pada lingkungan yang padat dengan jumlah 300 node, terjadi perbedaan selisih *end-to-end delay* sebesar 844.49 ms atau terjadi

penurunan *end-to-end delay* sebesar 73.50% antara routing protocol AODV asli dengan routing protocol AODV yang telah dimodifikasi, dimana routing protocol AODV yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika ketiga lingkungan tersebut dibandingkan, pada lingkungan yang sedang dan padat AODV modifikasi selalu lebih unggul dalam hal *end-to-end delay*. Tetapi apabila kita lihat selisih penurunan *End to End Delay* pada setiap skenario nya kita bisa melihat bahwa selisih kenaikan *End to End Delay* antara AODV dibandingkan dengan AODV yang telah dimodifikasi selalu meningkat dengan semakin padatnya jumlah node, hal ini disebabkan karena semakin padat node pada suatu skenario semakin banyak juga node yang bisa menjadi rute pengganti saat jumlah *routing table* pada suatu rutes menjadi padat, sehingga membuat selisih penurunan End to End Delay semakin tinggi. Hasil pengambilan data rata-rata untuk *forwarded route request* pada skenario *real* dengan jumlah *node* 60, 150, dan 300 dapat dilihat pada Gambar 5.8



Gambar 5.8 Grafik Forwarded Route Request Skenario Real

Berdasarkan grafik pada Gambar 5.8, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ

F) yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 1691.12, dimana terjadi penurunan *forwarded route request* sebesar 32,12% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal *forwarded route request* tersebut karena menghasilkan *forwarded route request* yang lebih rendah dari *routing protocol* AODV asli.

Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 3118.58, dimana terjadi penurunan *forwarded route request* sebesar 74.64% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam *forwarded route request* tersebut dari *forwarded route request* AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 3823.10, dimana terjadi penurunan *forwarded route request* sebesar 72.96% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal *forwarded route request* tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa pada lingkungan dengan node yang sedang, menghasilkan *forwarded route request* yang lebih sedikit daripada di lingkungan dengan jumlah node yang sedang jarang yang padat baik untuk routing protocol AODV murni maupun routing protocol AODV yang telah dimodifikasi.

Terjadi kenaikan yang sangat derastis pada skenario dengan menggunakan 60 *node* hal ini disebabkan karena pada saat jumlah node yang jarang di skenario real, persebaran node tidak merata sehingga diperlukan *forwarding request* berkali kali pada saat pengiriman RREQ dari *source* menuju *destination* yang akan menyebabkan peninkatan jumlah *forwarded route request.*

Nilai rata – rata penurunan yang terjadi pada *forwarded route request* adalah sebesar 58.79%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan *forwarded route request* yang lebih bagus atau dalam hal ini lebih rendah daripada *forwarded route request* asli dengan jumlah selisih *forwarded route request* yang cukup signifikan.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Pada Bab ini akan diberikan kesimpulan yang diperoleh dari Tugas Akhir yang telah dikerjakan dan saran tentang pengembangan dari Tugas Akhir ini yang dapat dilakukan di masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Dengan menambahkan perhitungan jumlah routing table pada *routing protocol* AODV dan membatasi *node* mana saja yang bisa me*rebroadcast* paket RREQ dengan menggunakan *threshold* dari routing table, maka dapat mengurangi jumlah *forwarding node* yang melakukan *rebroadcast*.
2. Dampak pembatasan *forwarding node* terhadap performa *routing protocol* AODV secara keseluruhan untuk skenario *real* menghasilkan peningkatan rata-rata *packet delivery ratio* sebesar 3.49%, rata-rata penurunan *routing overhead* sebesar 7.14%, rata-rata penurunan *end to end delay* sebesar 36.64%, dan juga rata-rata penurunan *forwarded route request* sebesar 58.79%.

6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. Untuk mendapatkan hasil uji coba yang lebih baik, dapat dilakukan uji coba yang lebih banyak, misal lebih dari 10 kali percobaan untuk tiap skenario.
2. Kedepannya dapat melakukan perhitungan jumlah *routing table* yang lebih dinamis pada *routing protocol* AODV dan untuk lingkungan yang dinamis, misalnya dapat menentukan *threshold routing table* secara dinamis dengan menggunakan

rumus yang dinamis sehingga perhitungan jumlah *node* tetangga lebih akurat.

3. Ide untuk implementasi pengurangan jumlah *forwarding node* sudah bagus, kedepannya bisa mengimplementasikan dengan menambahkan aspek lain yang dijadikan untuk pembatasan *forwarding node* seperti energi, kecepatan, arah, dan lain-lain.

DAFTAR PUSTAKA

[1] R. S. Raw, M. Kumar dan N. Singh, "SECURITY CHALLENGES, ISSUES AND THEIR," *International Journal of Network Security & Its Applications (IJNSA),* vol. 5, 2013.

[2] N. Chavhan dan A. Dhamgaye, "Survey on security challenges in VANET," *International Journal of Computer Science and Network,* vol. 2, no. 1, pp. 88-96, 2013.

[3] B. Paul dan M. Ibrahim, "VANET Routing Protocols: Pros and Cons," *International Journal of Computer Applications,* vol. 20, 2011.

[4] T. Kabir, N. Nurain dan M. H. Kabir, "Pro-AODV (Proactive AODV): Simple modifications to AODV for proactively minimizing congestion in VANETs," dalam *Networking Systems and Security (NSysS), 2015 International Conference on*, Dhaka, 2015.

[5] L. Khan, N. Ayub dan A. Saeed, "Anycast Based Routing in Vehicular Adhoc Networks (VANETS) using Vanetmobisim," COMSATS IIT Abbottabad, Pakistan, 2009.

[6] N. M. Mittal dan S. Choudhary, "Comparative Study of Simulators for Vehicular Ad-hoc Networks (VANETs)," IJETAE, Haryana, India, 2014.

[7] "OpenStreetMap," OpenStreetMap, [Online]. Available: https://www.openstreetmap.org/about. [Diakses 6 December 2017].

[8] "JOSM," JOSM, [Online]. Available: http://wiki.openstreetmap.org/wiki/JOSM. [Diakses 6 December 2017].

[9] "SUMO," SUMO, [Online]. Available: http://www.sumo.dlr.de/userdoc/Sumo_at_a_Glance.html. [Diakses 6 December 2017].

[10 A. V. Aho, B. W. Kerninghan dan P. J. Weinberger, The AWK Programming Language, Boston: Addison-Wesley, 1988.

[11 K. Majumder dan S. K. Sarkar, "Performance Analysis of AODV and DSR Routing Protocols in Hybrid Network Scenario," IEEE, Gujarat, India, 2009.

[12 M. Tamilarasi, S. S. V. R, U. M. Haputhantiri, C. Somathilaka, N. R. Babu, S. Chandramathi dan T. G. Palanivelu, "Scalability Improved DSR Protocol for MANETs," IEEE, Puducherry, 2007.

[13 S. Y. Dong, "Optimization of OLSR Routing Protocol in UAV ad Hoc Network," IEEE, Sichuan, 2016.

[14 S. N. Ferdous dan M. S. Hossain, "Randomized Energy-Based AODV Protocol For Wireless Ad-Hoc Network," IEEE, Dhaka, 2016.

[15 mahbubulalam, "mahbubulalam," [Online]. Available: http://mahbubulalam.com/convergence-of-secure-vehicular-ad-hoc-network-and-cloud-in-iot/. [Diakses 30 May 2018].

LAMPIRAN

A.1 Kode Fungsi AODV::nb_insert

```
void
AODV::nb_insert(nsaddr_t id) {
countNeib[index] = countNeib[index] + 1;
AODV_Neighbor *nb = new
AODV_Neighbor(id);

 assert(nb);
 nb->nb_expire = CURRENT_TIME +
                 (1.5 * ALLOWED_HELLO_LOSS
* HELLO_INTERVAL);
 LIST_INSERT_HEAD(&nbhead, nb, nb_link);
 seqno += 2;             // set of
neighbors changed
 assert ((seqno%2) == 0);
}


AODV_Neighbor*
AODV::nb_lookup(nsaddr_t id) {
AODV_Neighbor *nb = nbhead.lh_first;

 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id) break;
 }
 return nb;
}
```

A.2 Kode Fungsi AODV::nb_delete

```
void
AODV::nb_delete(nsaddr_t id) {
countNeib[index] = countNeib[index] - 1 ;

AODV_Neighbor *nb = nbhead.lh_first;

 log_link_del(id);
 seqno += 2;     // Set of neighbors
changed
 assert ((seqno%2) == 0);

 for(; nb; nb = nb->nb_link.le_next) {
   if(nb->nb_addr == id) {
     LIST_REMOVE(nb,nb_link);
     delete nb;
     break;
   }
 }

 handle_link_failure(id);

}
```

A.3 Kode Fungsi AODV::rt_update

```
void
AODV::rt_update(aodv_rt_entry *rt,
u_int32_t seqnum, u_int16_t metric,
                nsaddr_t nexthop, double
expire_time) {
countRoute[index] = countRoute[index] +
1;

     rt->rt_seqno = seqnum;
     rt->rt_hops = metric;
     rt->rt_flags = RTF_UP;
     rt->rt_nexthop = nexthop;
     rt->rt_expire = expire_time;
     rtable.rt_display(index);
}
```

A.4 Kode Fungsi AODV::rt_down

```
void
AODV::rt_down(aodv_rt_entry *rt) {


  /*
   *  Make sure that you don't "down" a
route more than once.
   */

  if(rt->rt_flags == RTF_DOWN) {
    return;
  }
  countRoute[index] = countRoute[index] -
1;

  // assert (rt->rt_seqno%2); // is the
seqno odd?
  rt->rt_last_hop_count = rt->rt_hops;
  rt->rt_hops = INFINITY2;
  rt->rt_flags = RTF_DOWN;
  rt->rt_nexthop = 0;
  rt->rt_expire = 0;

}
```

A.5 Kode Fungsi AODV::recvRequest

```
void
AODV::recvRequest(Packet *p) {
//modified AODV calculate neighbor
  Node* m_node = Node::get_node_by_address(this-
>addr());
  neighbor_list_node* my_mobile_neighbor_list;
  my_mobile_neighbor_list = m_node->neighbor_list_;
  int count = 0;

  //cout<<"Node id : " <<index<<endl;
  while(my_mobile_neighbor_list)
  {
      //cout<<"Neighbor ID:"<<my_mobile_neighbor_list-
>nodeid<<endl;
      count++;
      if(my_mobile_neighbor_list->next){

my_mobile_neighbor_list=my_mobile_neighbor_list->next;
      }
      else{
          //cout<<"## Jumlah: "<<count<<endl;
          //cout<<"## Jumlah: "<<nb_node()<<endl;
          break;
      }
  }

struct hdr_ip *ih = HDR_IP(p);
struct hdr_aodv_request *rq = HDR_AODV_REQUEST(p);
aodv_rt_entry *rt;


  /*
   * Drop if:
   *      - I'm the source
   *      - I recently heard this request.
   */
```

```
if(rq->rq_src == index) {
#ifdef DEBUG
    fprintf(stderr, "%s: got my own REQUEST\n",
__FUNCTION__);
#endif // DEBUG

    Packet::free(p);
    return;
  }

 if (id_lookup(rq->rq_src, rq->rq_bcast_id)) {

#ifdef DEBUG
   fprintf(stderr, "%s: discarding request\n",
__FUNCTION__);
#endif // DEBUG


   Packet::free(p);
   return;
 }
 ////////////////MODIFIKASI AODV
  if (countNeib[index] > 18 && countRoute[index] > 21
&& rq->rq_dst != index){


     Packet::free(p);
     return;
     //cout<<"Drop paket"<<endl;
   }
//////////////////////////
 /*
  * Cache the broadcast ID
  */
 id_insert(rq->rq_src, rq->rq_bcast_id);
aodv_rt_entry *rt0;
rt0 = rtable.rt_lookup(rq->rq_src);
   if(rt0 == 0) { /* if not in the route table */
   // create an entry for the reverse route.
     rt0 = rtable.rt_add(rq->rq_src);
   }

   rt0->rt_expire = max(rt0->rt_expire, (CURRENT_TIME +
REV_ROUTE_LIFE));
```

```
   if ( (rq->rq_src_seqno > rt0->rt_seqno ) ||
        ((rq->rq_src_seqno == rt0->rt_seqno) &&
         (rq->rq_hop_count < rt0->rt_hops)) ) {
   // If we have a fresher seq no. or lesser #hops for
the
   // same seq no., update the rt entry. Else don't
bother.
rt_update(rt0, rq->rq_src_seqno, rq->rq_hop_count, ih-
>saddr(),
               max(rt0->rt_expire, (CURRENT_TIME +
REV_ROUTE_LIFE)) );
     if (rt0->rt_req_timeout > 0.0) {
     // Reset the soft state and
     // Set expiry time to CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT
     // This is because route is used in the forward
direction,
     // but only sources get benefited by this change
       rt0->rt_req_cnt = 0;
       rt0->rt_req_timeout = 0.0;
       rt0->rt_req_last_ttl = rq->rq_hop_count;
       rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
     }

     /* Find out whether any buffered packet can
benefit from the
      * reverse route.
      * May need some change in the following code -
Mahesh 09/11/99
      */
     assert (rt0->rt_flags == RTF_UP);
     Packet *buffered_pkt;
     while ((buffered_pkt = rqueue.deque(rt0->rt_dst)))
{
       if (rt0 && (rt0->rt_flags == RTF_UP)) {
       assert(rt0->rt_hops != INFINITY2);
         forward(rt0, buffered_pkt, NO_DELAY);
       }
     }
   }
```

```
rt = rtable.rt_lookup(rq->rq_dst);

 // First check if I am the destination ..

 if(rq->rq_dst == index) {

#ifdef DEBUG
   fprintf(stderr, "%d - %s: destination sending
reply\n",
                   index, __FUNCTION__);
#endif // DEBUG
   seqno = max(seqno, rq->rq_dst_seqno)+1;
   if (seqno%2) seqno++;

   sendReply(rq->rq_src,           // IP Destination
             1,                    // Hop Count
             index,                // Dest IP Address
             seqno,                // Dest Sequence Num
             MY_ROUTE_TIMEOUT,     // Lifetime
             rq->rq_timestamp);    // timestamp

   Packet::free(p);
 }
else if (rt && (rt->rt_hops != INFINITY2) &&
                (rt->rt_seqno >= rq->rq_dst_seqno) ) {

   //assert (rt->rt_flags == RTF_UP);
   assert(rq->rq_dst == rt->rt_dst);
   //assert ((rt->rt_seqno%2) == 0);  // is the seqno
even?
   sendReply(rq->rq_src,
             rt->rt_hops + 1,
             rq->rq_dst,
             rt->rt_seqno,
             (u_int32_t) (rt->rt_expire - CURRENT_TIME),
             //             rt->rt_expire -
CURRENT_TIME,
             rq->rq_timestamp);
   // Insert nexthops to RREQ source and RREQ
destination in the
   // precursor lists of destination and source
respectively
   rt->pc_insert(rt0->rt_nexthop); // nexthop to RREQ
source
   rt0->pc_insert(rt->rt_nexthop); // nexthop to RREQ
destination
```

```
#ifdef RREQ_GRAT_RREP

   sendReply(rq->rq_dst,
             rq->rq_hop_count,
             rq->rq_src,
             rq->rq_src_seqno,
            (u_int32_t) (rt->rt_expire - CURRENT_TIME),
            //             rt->rt_expire -
CURRENT_TIME,
             rq->rq_timestamp);
#endif
        Packet::free(p);
 }
 /*
  * Can't reply. So forward the  Route Request
  */
 else {
   ih->saddr() = index;
   ih->daddr() = IP_BROADCAST;
   rq->rq_hop_count += 1;
   // Maximum sequence number seen en route
   if (rt) rq->rq_dst_seqno = max(rt->rt_seqno, rq-
>rq_dst_seqno);
   forward((aodv_rt_entry*) 0, p, DELAY);
 }

}
```

A.6 Kode Skenario NS-2

```
set val(chan)   Channel/WirelessChannel;
set val(prop)   Propagation/TwoRayGround;
set val(netif)  Phy/WirelessPhy;
set val(mac)    Mac/802_11;
set val(ifq)    Queue/DropTail/PriQueue;
set val(ll)     LL;
set val(ant)    Antenna/OmniAntenna;
set opt(x)      1300;
set opt(y)      1300;
set val(ifqlen) 1000;
set val(nn)     60;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)      "cbr1.txt";
set val(sc)      "scen1modif.txt";

set ns_          [new Simulator]

# setup topography object

set topo   [new Topography]

# create trace object for ns and nam

set tracefd     [open scenario1.tr w]
set namtrace    [open scenario1.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace
$opt(x) $opt(y)
```

```
# Create God
set god_ [create-god $val(nn)]

#global node setting
$ns_ node-config -adhocRouting
$val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan)
\
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \

# 802.11p default parameters
Phy/WirelessPhy set  RXThresh_ 5.57189e-
11 ; #400m
Phy/WirelessPhy set  CSThresh_ 5.57189e-
11 ; #400m

#  Create the specified number of nodes
[$val(nn)] and "attach" them
#  to the channel.
for {set i 0} {$i < $val(nn)} {incr i} {
          set node_($i) [$ns_ node]
          $node_($i) random-motion 0 ;#
disable random motion
}
```

```
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam,
must adjust it according to your scenario
    # The function must be called after
mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i)
reset";
}

#$ns_ at  $val(stop)  "stop"
$ns_ at  $val(stop).0002 "puts \"NS
EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x
$opt(x) y $opt(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp
$val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant
$val(ant)"

puts "Starting Simulation..."
$ns_ run
```

A.7 Kode Konfigurasi *Traffic*

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"


set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(56) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(57) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 4.5568388786897245  "$cbr_(0) start"
```

A.8 Kode Skrip AWK *Packet Delivery Ratio*

```
BEGIN {
        sendLine = 0;
        recvLine = 0;
        fowardLine = 0;
}

$0 ~/^s.* AGT/ {
        sendLine ++ ;
}

$0 ~/^r.* AGT/ {
        recvLine ++ ;
}

$0 ~/^f.* RTR/ {
        fowardLine ++ ;
}

END {
        printf "cbr s:%d r:%d, r/s
Ratio:%.4f, f:%d \n", sendLine, recvLine,
(recvLine/sendLine),fowardLine;
}
```

A.9 Kode Skrip AWK Rata-Rata *End-to-End Delay*

```
BEGIN{
      sum_delay = 0;
      count = 0;
}
{
      if ($2 >= 101) {

            if($4 == "AGT" && $1 == "s" &&
seqno < $6) {
              seqno = $6;
            }

            if($4 == "AGT" && $1 == "s") {
                  start_time[$6] = $2;
            }

            else if(($7 == "cbr") && ($1 ==
"r")) {
                  end_time[$6] = $2;
            }

            else if($1 == "D" && $7 == "cbr")
{
                  end_time[$6] = -1;
            }
      }
}
END {

    for(i=0; i<=seqno; i++) {
            if(end_time[i] > 0) {
                  delay[i] = end_time[i] -
start_time[i];
                  count++;
        }
        else {
            delay[i] = -1;
        }
    }
```

```
    for(i=0; i<=seqno; i++) {
        if(delay[i] > 0) {
           n_to_n_delay = n_to_n_delay +
delay[i];
        }
    }
     n_to_n_delay = n_to_n_delay/count;
     printf "End-to-End Delay \t= "
n_to_n_delay * 1000 " ms \n";
}
```

A.10 Kode Skrip AWK *Routing Overhead*

```
BEGIN {
      rt_pkts = 0;
}
{
      if ($2 >= 101) {
            if (($1 == "s" || $1 == "f")
&& ($4 == "RTR") && ($7 == "AODV")) {

                  rt_pkts++;
            }
      }
}
END {
      printf "Routing Packets \t= %d \n",
rt_pkts;
}
```

A.11 Kode Skrip AWK *Forwarded Route Request*

```
BEGIN {
      rreqf = 0;
}
{
      if ($2 >= 101) {
            if (($1 == "f") && ($4 ==
"RTR") && ($7 == "AODV") ($19 == "[1")) {

                  rreqf++;
      }
}
END {
      printf "Route Request F \t= %d \n",
rreqf;
}
```

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Faiq Firdausy lahir di Surabaya pada 24 Mei 1996. Penulis menempuh pendidikan formal TK Liya (2001-2002), SDN Ketabang 1 (2002-2008), SMPN 9 Surabaya (2008-2011), SMAN 5 Surabaya (2011-2014), dan melanjutkan studi S1 di Departemen Informatika ITS (2014-2018). Bidang studi yang diambil oleh penulis pada saat berkuliah di Departemen Informatika ITS adalah Arsitektur Jaringan Komputer (AJK). Penulis aktif dalam organisasi Himpunnan Mahasiswa Teknik Computer-Informatika (2015-2017). Penulis juga aktif dalam kegiatan kepanitian seperti SCHEMATICS 2015 dan SCHEMATICS 2016 divisi Hubungan Masyarakat (Humas). Penulis pernah kerja praktik di PT. Pertamina (Persero) periode Juli – Agustus 2017. Penulis dapat dihubungi melalui nomor *handphone* 085655237943 atau di email faiqfirdausy16@gmail.com.